Centre de recherche sur les transports (C.R.T.)
Centre for Research on Transportation

CRT-2004-02

AUTOMATISATION DE LA PARALLÉLISATION DE SYSTÈMES COMPLEXES AVEC APPLICATION À L'ENVIRONNEMENT MATLAB/SIMULINK

par

Abdessamad Ait El Cadi

Mémoire de maîtrise

Janvier 2004

Université de Montréal
C.P. 6128, succursale Centre-ville, Montréal QC H3C 3J7 Canada
Téléphone : (514) 343-7575 / Télécopieur : (514) 343-7121

crt@crt.umontreal.ca
www.crt.umontreal.ca

HEC ÉCOLE POLYTECHNIQUE MONTRÉAL Université de Montréal

104-02684/

PUBLICATION
CRT-2004-02

AUTOMATISATION DE LA PARALLÉLISATION DE SYSTÈMES COMPLEXES AVEC APPLICATION À L'ENVIRONNEMENT MATLAB/SIMULINK

par

Abdessamad Ait El Cadi[1]

Mémoire de maîtrise présenté en vue de l'obtention du diplôme de M.Sc. appliquées (Génie industriel). Directrice de recherche : Sophie Lapierre.

[1] Département de mathématiques et de génie industriel, École Polytechnique et Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Canada H3C 3J7

Dépôt légal – Bibliothèque nationale du Québec
Bibliothèque nationale du Canada, 2004

Ce travail est dédié à ma famille que j'adore;

La famille Ait El Cadi.

Mais tout spécialement à mon ange gardien;

Ma chère maman, Mme Fattouma Bouchoua.

Comme je le dédie aussi à ma chère directrice Sophie Lapierre pour son courage

et pour sa personne.

o0o

REMERCIEMENTS

Je remercie M. Pierre Baptiste, professeur à l'École Polytechnique de Montréal, qui m'a fait l'honneur de présider ce jury.

Je remercie ma directrice, Mme. Sophie Lapierre et M. Teodor Gabriel Crainic, qui m'ont tant soutenu par leurs conseils, leurs suggestions pertinentes et surtout par leur présence. Merci du fond du coeur.

Je remercie l'Agence Spatiale Canadienne ainsi que l'équipe d'Opal-rt pour leur soutien matériel et financier et pour leur esprit d'équipe.

Je remercie l'École Polytechnique de Montréal, le Centre de Recherche sur le Transport et mon école l'École Nationale des Ponts et Chaussées pour m'avoir accordé conjoitement cette chance de visiter ce pays merveilleux et de réaliser ce mémoire.

Je remercie tous mes colocs (surtout mes coloques) et tous mes ami(e)s pour être toujours là, pour avoir mis un brin de lumière dans ma vie pour m'avoir conseillé, chicané, assisté, soutenu ... Pour tout et je vous dis JE VOUS ADORE!

Tous mes remerciements aussi à la ville de Montréal au Québec et à tous les Québecois.

RÉSUMÉ

La simulation et l'étude de systèmes avancés tels que des robots requièrent une puissance accrue de calcul que les processeurs séquentiels sont, actuellement, loin de satisfaire. Afin de pallier à cette lacune, le calcul parallèle est une solution pour réduire les temps de calcul. En faisant collaborer des processeurs indépendants, nous pouvons augmenter notre capacité de calcul et, par la suite, répondre à de telles exigences. Le présent projet, fait en collaboration avec Opal-rt technologies et l'Agence Spatiale Canadienne, propose une solution pratique pour automatiser la parallélisation de simulations développées sous l'environnement Matlab/Simulink. La solution a été développée en deux phases : génération du graphe des tâches, puis l'affectation des tâches entre les processeurs. Notre étude de faisabilité est concluante et il reste encore des travaux de recherche à effectuer avant la phase de commercialisation.

Matlab/Simulink permet de développer des simulations à l'aide d'un langage de schématisation fonctionnel. La logique de ce type de langage permet déjà une élaboration bien structurée du code. Toutefois, avant de pouvoir automatiser la parallélisation, il faut formaliser le modèle Simulink sous forme d'un graphe pour s'assurer que les tâches sont bien identifiées et que les contraintes de précédences et de communication sont justement représentées. Pour ce faire, nous avons recours à la génération d'un graphe acyclique orienté (DAG). La génération d'un tel graphe est tout un défi. Premièrement, le modèle réel contient des cycles et les blocs sont regroupés de manière hiérarchique en différents niveaux. Deuxièmement, il faut rendre cette procédure automatique pour n'importe quel type de modèle. La solution que nous avons développée est basée sur les outils de Matlab. Nous avons créé un quasi-compilateur de Simulink qui décortique le modèle pour détecter toutes ses composantes. Ce compilateur permet, pour un niveau de détail donné ou "granu-

larité voulue", de déterminer de manière automatique les tâches et leurs liens. En outre, une bonne caractérisation des boucles dans le modèle nous permet de les traiter adéquatement et ainsi éviter les cycles dans le graphe généré.

Une fois le graphe acyclique généré, nous devons affecter les tâches aux différents CPUs tout en respectant les contraintes de précédence et les délais de communication en visant un temps total d'exécution le plus court possible. Nous avons parcouru la littérature relative à ce sujet. Il existe plusieurs méthodes de résolution mais, vu la nature du projet - étude de faisabilité - nous avons implanté ce qui est le plus simple et le plus sûr soit les heuristiques de construction par règle de priorité et ceux consistants à regrouper les tâches (méthode de "Clustering"). Nous ne permettons pas la duplication des tâches mais avons brièvement exploré le potentiel de celle-ci. Nous avons appliqué ces deux types d'heuristiques à notre problème. Adapter ces algorithmes à notre situation a toutefois été plus difficile que prévu. L'implantation du code a donné, après plusieurs améliorations, les résultats escomptés.

Pour valider nos résultats, nous avons mené plusieurs groupes de tests. D'une part, un groupe de tests concerne un exemple pratique de simulation d'un robot. Ces tests ont pour objectifs de valider notre méthode dans un contexte réel et de s'assurer de l'intégrité du modèle après l'automatisation de la séparation. En d'autres termes justifier la faisabilité du projet. D'autre part, un autre groupe de tests a été effectué sur un ensemble représentatif de graphes pour tester la robustesse, la stabilité et la performance de notre heuristique lorsque nous n'avons pas le contrôle pratique de l'ordonnanceur. Les résultats de ces tests sont très concluants : notre heuristique fournit de bonnes solutions et les meilleures de ces dernières dans un contexte contrôlé restent toujours performantes lorsque l'on ne contrôle pas l'ordonnanceur, ce qui s'avère le cas de l'environnement Rt-lab.

L'automatisation de la parallélisation à partir d'un modèle schématique (du type Simulink/Matlab) est donc possible. Nos heuristiques donnent des solutions acceptables. Toutefois, nous soupçonnons que l'ajout de métaheuristiques pourrait augmenter la qualité des solutions car nous obtenons des solutions moins bonnes lorsque le DAG est généré avec une plus fine granularité. Le mémoire est donc concluant sur le plan pratique mais il faut poursuivre les travaux de recherche pour commercialiser le produit.

Mots clefs : simulation distribuée, calcul parallèle, Simulink, ordonnancement, heuristique avec règles de priorité, clustering, délais de communication.

ABSTRACT

The simulation and the analysis of advanced systems such as robots require an increased design power, which sequential processors are far from satisfying. Parallel computing is a solution to reduce the execution times. By using independent processors together, we can increase our capacity of calculation and answer such requirements thereafter. In this project, made in collaboration with Opal-rt technologies and the Canadian Space Agency, we propose a practical solution to automate the parallelization of simulations developed under the Matlab/Simulink environment. The solution was developed in two phases: generation of the task graph, then the task assignment to the processors. Our feasibility study is conclusive and a little research work remains to be performed before launching our software on the market.

Under Matlab/Simulink, we can develop simulations using a functional bloc-diagram language. The logic of this kind of language already allows a well-structured code. However, to automate parallelization, it is necessary to formalize the model in the form of a graph to make sure that the tasks are well identified and that the constraints of precedence and communication delays are well defined. That is why we generate a directed acyclic graph (DAG). The generation of such a graph is a big challenge. Firstly, the real model contains cycles and the blocks are gathered in a hierarchical way with various levels. Secondly, it is necessary to make this procedure automatic for any kind of models. The solution that we developed is based on the Matlab tools. We created Simulink compiler which surfs the model to detect all its components. This compiler allows, for a given level of details or for a " desired granularity ", to determine in an automatic way the tasks and their links. Moreover a good characterization of the loops in the model enables us to avoid the cycles in the generated graph.

Once the DAG is generated, we must assign the tasks to the different CPUs with respect to the communication delays and the precedence constraints while aiming the shortest possible execution time over all CPUs - which means the objective is to minimize the makespan. We performed a literature review on this subject. There are several methods of resolving this problem but, due to the nature of our project - a feasibility study - we began with what is simplest and rather sure, a priority-based heuristic and a "Clustering" algorithm. We do not allow task duplication but we explore it. We applied these types of heuristic to our problem. Adapting these algorithms to our situation was more difficult than expected. The implementation of the code gave, after many improvements, good results - as good as we had planned.

To validate our results we performed several tests. On the one hand, a group of tests relates to a realistic example of a robot simulation. This test aims to validate our method in a real context and to ensure the accuracy of the results after the automation of the whole process. In other words, these tests prove the feasibility of the project. On the other hand, another group of tests use a group of representative graphs to test the robustness, the stability and the performance of our heuristics in cases when there is no practical control of the scheduler. The results of these tests are conclusive. Our heuristics provides good solutions and the best schedules in a controlled context are still the best when the scheduler is not under control, which is the case of the Rt-lab environment.

This proves that the automation of the parallelization of a schematic model, the type of the Simulink/Matlab models, is possible. The heuristics give acceptable results. However, due to the fact that we obtain worse solutions when the DAG is generated with a finer granularity, we suspect that metaheuristics could increase the quality of the solutions. Also, while the report is conclusive on the practical

level, it is necessary to continue the research tasks before delivering the product to the market.

Keywords: distributed simulation, parallel computing, Simulink, scheduling, priority-based heuristic, clustering, communication delays.

TABLE DES MATIÈRES

DÉDICACE iv

REMERCIEMENTS v

RÉSUMÉ vi

ABSTRACT ix

TABLE DES MATIÈRES xii

LISTE DES TABLEAUX xvi

LISTE DES FIGURES xviii

LISTE DES ANNEXES xxii

INTRODUCTION 1

CHAPITRE 1 : PRÉSENTATION ET MISE EN CONTEXTE DU PROBLÈME 4

1.1 L'environnement informatique de Rt-lab 5

1.1.1 Conception graphique 6

1.1.2 Matlab 7

1.1.3 Simulink 8

1.1.4 Présentation du logiciel Rt-lab 10

1.1.5 Procédure de parallélisation sous Rt-lab 13

1.1.6 Le langage bloc de Simulink 14

1.2 La problèmatique 19

CHAPITRE 2 : REVUE DE LA LITTÉRATURE 21
2.1 Les différentes approches de parallélisation 22
2.2 La génération du graphe des tâches 24
2.3 Formulation mathématique du problème 25
2.4 Méthodes de résolution 26
2.4.1 Les heuristiques 26
2.4.1.1 Les heuristiques basées sur des règles de priorité 27
2.4.1.2 "Clustering" 28
2.4.2 Les Métaheuristiques 29
2.4.2.1 La recherche avec tabous 29
2.4.2.2 Les algorithmes génétiques 30
2.4.2.3 Le recuit simulé 31
2.5 Conclusion 32

CHAPITRE 3 : GÉNÉRATION DU GRAPHE ACYCLIQUE 33
3.1 Aspects logiques 33
3.1.1 Des blocs et flèches de Simulink vers les noeuds et les arcs du DAG . 36
3.1.1.1 Blocs et tâches - noeuds 37
3.1.1.2 Flèches et liens d'antécédence - arcs 37
3.1.2 Granularité du graphe 38
3.1.3 Modèle cyclique 40
3.1.3.1 Origine et nature du caractère cyclique 42
3.1.3.2 Modélisation proposée pour rendre le graphe acyclique 44
3.1.3.3 La représentation des phénomènes cycliques dans Simulink 46
3.1.3.4 Le fonctionnement des blocs "Non-Feedthrough" 48
3.1.3.5 Représentation des blocs "Non-Feedthrough" 49
3.1.3.6 Conclusion concernant le graphe acyclique 51
3.2 Aspects techniques de l'automatisation 52

3.2.1 Description de l'algorithme de génération du DAG 52
3.2.1.1 Caractérisation des tâches et des liens d'antécédence 53
3.2.1.2 Génération du DAG - de la matrice d'adjacence 57
3.2.1.3 Création de la table des attributs des tâches 59
3.2.2 Exemple d'application de l'algorithme de Génération du DAG 60

CHAPITRE 4 : RÉSOLUTION DU PROBLÈME DE PARALLÉLISATION 64
4.1 Le problème de parallélisation 64
4.2 Modélisation mathématique 66
4.2.1 Notation 67
4.2.2 Modèle horaire 68
4.2.2.1 Cas d'une architecture homogène 70
4.2.2.2 Cas d'une architecture homogène et de mémoire partagée 71
4.2.3 Modèle par séquence 72
4.3 Algorithme de résolution 74
4.3.1 Logique des heuristiques basées sur la règle de priorité 74
4.3.2 Exemple d'application 76
4.3.3 L'algorithme 81
4.3.3.1 Programme principal 82

CHAPITRE 5 : TESTS ET ANALYSE DES RÉSULTATS 86
5.1 Tests de performance 87
5.1.1 Objectif du test 87
5.1.2 Les données pour le test 87
5.1.3 Le protocole du test 90
5.1.3.1 Les différents scénarios à tester 91
5.1.3.2 Les données à collecter et les mesures de performance 91
5.1.4 Les résultats du test 92
5.1.5 Analyse des résultats. 93

5.2 Tests pratiques 99
5.2.1 Objectif du test 100
5.2.2 Objet du test 100
5.2.3 Résultats et conclusion 101
5.3 Tests de robustesse 107
5.3.1 Objectif du test 108
5.3.2 Les données pour le test 108
5.3.3 Déroulement du test 108
5.3.4 Les résultats du test 109
5.3.5 Conclusion 110

CONCLUSION 112

RÉFÉRENCES 116

ANNEXES 123

LISTE DES TABLEAUX

Tableau 5.1 Description du jeu de graphes utilisés pour le test de performance. 89

Tableau 5.2 Données contenu dans le fichier résultat du TestManager . . 92

Tableau 5.3 Facteur d'accélération "SpeedUp" selon le type de communication, la stratégie d'affectation et la règle de priorité pour les graphes-tests 93

Tableau 5.4 Pourcentage d'utilisation du CPU 1 selon le type de communication, la stratégie d'affectation et la règle de priorité pour les graphes-tests 94

Tableau 5.5 Pourcentage d'utilisation du CPU 2 selon le type de communication, la stratégie d'affectation et la règle de priorité pour les graphes-tests 94

Tableau 5.6 Facteur d'accélération moyen "SpeedUp" pour chaque graphe-test et règle de priorité. 95

Tableau 5.7 Temps d'exécution et facteur d'accélération par modèle et par solution (le M1. fait référence à des variantes du robot à deux degrés de liberté et le modèle M2 fait référence au système des deux manipulateurs. Le C_{max} est indiqué en gras.) 104

Tableau 5.8 Résultat pour une séparation sur 3 CPU pour le modèle M1.5 (Robot à deux degrés de liberté). Le C_{max} est indiqué en gras. 104

Tableau 5.9 Écart entre les résultats prédits par l'heuristique et les valeurs obtenu réellement. 106

Tableau 5.10 Moyenne des écarts relatifs entre le facteur d'accélération prédit et les valeurs obtenus par le simulateur. 109

Tableau 5.11 Intervalle de confiance à 95% sur les moyennes de la table 5.10. 110

Tableau II.1 Parameters of DAG-generation and heuristic. 147

Tableau II.2 Distribution obtained with heuristic solution. 147
Tableau II.3 Simulation parameters. 148

LISTE DES FIGURES

Figure 1.1	Exemple d'un modèle sous Simulink (calcul du minimum de trois nombres)	9
Figure 1.2	Le modèle de la figure 1.1 éclaté	9
Figure 1.3	L'Architecture de Rt-lab	11
Figure 1.4	Mécanisme de RTW(Real-Time Workshop)	12
Figure 1.5	Procédure manuelle de séparation	15
Figure 1.6	Ensemble des fichiers générés par Rt-lab pour le modèle "exemple.mdl"	16
Figure 1.7	Le fichier ".mdl" du moèle de la figure 1.1	17
Figure 1.8	Le fichier ".c" représentant le code généré par RTW pour le modèle Simulink de la figure 1.7 (et 1.1)	18
Figure 3.1	Exemple de modèle Simulink représentant un robot de deux degrés de liberté - un bras avec deux articulations.	34
Figure 3.2	Le DAG correspondant au modèle Simulink de la figure 3.1.	35
Figure 3.3	Exemple de correspondance entre Simulink et le DAG. Les blocs d'affichage et d'initialisation "Constant4" et "Scope" ne sont pas considérés comme des tâches.	36
Figure 3.4	Exemple de correspondance entre Simulink et le DAG après éclatement des sous-systèmes.	40
Figure 3.5	Exemple d'un faux modèle cyclique (le cycle est formé entre les blocs "Subsystem" et "Switch"). Le rectangle en bas montre le contenu du sous-système "Subsystem".	41
Figure 3.6	Le modèle de la figure 3.5 une fois éclaté et ramené à un seul niveau. On constate l'absence de la boucle cyclique	43
Figure 3.7	Exemple d'un système rétroactif contenant un cycle.	43
Figure 3.8	Exemple d'un vrai modèle cyclique : un pendule simple.	44

Figure 3.9	Un exemple de cycle	45
Figure 3.10	Un exemple de traitement d'un cycle. Le lien 4 – 1 est retardé.	46
Figure 3.11	Exemple de blocs "FeedThrough" (Gain) et "Non-FeedThrough" (Integrateur).	48
Figure 3.12	Exemple de modèle cyclique avec des tâches "Non-Feedthrough" (les tâches grisées).	49
Figure 3.13	Le modèle équivalent à l'exemple de la figure 3.12 du point de vue de l'exécution sous Rt-lab.	50
Figure 3.14	Le modèle équivalent à l'exemple de la figure 3.7 du point de vue de l'exécution sous Rt-lab. avec l'ordre d'exécution des tâches (il est indiqué par les chiffres en haut et à droite des blocs.)	51
Figure 3.15	Algorithme de génération du graphe "DAG"	54
Figure 3.16	Pseudocode pour la caractérisation des tâches et des liens d'antécédence.	56
Figure 3.17	Pseudocode pour la génération de la matrice d'adjacence. .	58
Figure 3.18	Pseudocode pour la génération de la table des tâches. . . .	60
Figure 3.19	Le modèle Simulink "test-DAG"	61
Figure 3.20	La matrice d'adjacence correspondant au modèle de la figure 3.19.	61
Figure 3.21	La table des tâches correspondant au modèle de la figure 3.19.	62
Figure 3.22	Le DAG résultant des figures 3.19, 3.20 et 3.21.	63
Figure 4.1	Exemple d'ordonnancement des tâches dans le temps. . . .	65
Figure 4.2	Modèle Simulink "test-heuristique".	76
Figure 4.3	DAG du modèle Simulink de la figure 4.2.	77
Figure 4.4	Matrice d'adjacence issue du DAG de la figure 4.2.	77
Figure 4.5	Liste des tâches avec leurs durées issue du DAG de la figure 4.2.	78

Figure 4.6	Liste des tâches avec leurs priorités respectives calculées par l'algorithme de "Dijkstra" à partir du DAG de la figure 4.2.	78
Figure 4.7	Liste des tâches avec leurs priorités triées selon un ordre décroissant.	78
Figure 4.8	Les tâches disponibles avant le début de l'affectation.	79
Figure 4.9	Affectation de la tâche disponible "1" au CPU 1.	80
Figure 4.10	Les tâches disponibles après la première affectation.	81
Figure 4.11	Résultat de la séparation du modèle Simulink de la figure 4.2 sur deux CPUs.	82
Figure 5.1	Variation du "SpeedUp" moyen selon la divergence du graphe sur les graphes-tests avec la priorité Dijkstra (communication nulle).	96
Figure 5.2	Variation du "SpeedUp" moyen selon la convergence du graphe sur les graphes-tests avec la priorité Dijkstra (communication nulle).	97
Figure 5.3	Variation du "SpeedUp" moyen selon le nombre de tâches du graphe sur les graphes-tests avec la priorité Dijkstra (communication nulle).	98
Figure 5.4	Variation du "SpeedUp" moyen selon la densité du graphe sur les graphes-tests avec la priorité Dijkstra (communication nulle).	98
Figure 5.5	(a) Bras à deux degrés de liberté; (b) l'actionneur	101
Figure 5.6	Niveau supérieur du modèle Simulink du robot à deux degrés de liberté de la figure 5.5.	102
Figure 5.7	Deux manipulateurs exécutant une tâche de contact.	102
Figure 5.8	Niveau supérieur du modèle Simulink du robot de la figure 5.7.	103

Figure 5.9 Comparaison, pour le robot à deux degrés de liberté, entre les résultats prédits par l'heuristique et les résultats réels une fois le modèle séparé est exécuté sur Rt-lab. 105
Figure II.1 Proposed parallelization technique 133
Figure II.2 Mass-spring-damper system in (a) block-diagram form and (b) DAG representation 134
Figure II.3 Schematics of a 2-CPU simulation 139
Figure II.4 Robotic system 140
Figure II.5 Schematics of a 2-CPU simulation with direct feedthrough . 143
Figure II.6 Schematics of a 2-CPU simulation without a direct feedthrough 145
Figure II.7 Distributed simulation set-up 146
Figure II.8 (a) Joint angles vs. time, (b) joint speeds vs. time, and (c) joint speed errors vs. time 149

LISTE DES ANNEXES

ANNEXE I LES DONNÉES ET LES RÉSULTATS DES TESTS . . . 123

ANNEXE II ARTICLE : "PARALLÉLISATION AUTOMATIQUE DE SYSTÈMES ÉLECTRO-MÉCANIQUES BASÉS SUR DES ÉQUATIONS DIFFÉRENTIELLES ORDINAIRES" . . . 124

INTRODUCTION

Le progrès fulgurant des technologies durant les dernières décennies a eu un grand impact sur la conception, le design et l'ingénierie de systèmes de contrôle avancés tels les robots et les moteurs contrôlés électriquement. La simulation est généralement le moyen parfait pour aider à optimiser ces systèmes. Mais ces méthodes d'optimisation exigent des capacités de calcul énormes pour atteindre le niveau de précision désirée de conception. En dépit des améliorations récentes de vitesse de traitement, les ordinateurs d'aujourd'hui sont loin de fournir la rapidité de calcul suffisante pour de telles simulations. Ainsi, pour permettre l'essor de cette discipline qu'est le design de systèmes avancés, on a recours à des simulations exécutées sur plusieurs ordinateurs qui travaillent en collaboration. Cet environnement informatique est appelé le *calcul parallèle*. Il s'agit de traitement permettant l'exécution de plusieurs opérations simultanément sur un ordinateur à plusieurs processeurs ou sur plusieurs ordinateurs.

Il y a deux façons d'aborder la parallélisation des calculs pour la simulation. La première, la plus simple, consiste à répartir les différentes itérations sur différents ordinateurs. c.-à-d. chaque machine s'occupe du calcul d'un pas de simulation. La deuxième s'attaque au coeur du programme: elle vise à répartir les entités d'un programme sur plusieurs ordinateurs afin de réduire le temps total d'exécution. Cette dernière façon de faire, lorsque maîtrisée, permet vraiment de réduire les temps de calcul et facilite la conception de systèmes en temps réel. Cependant cette méthode est fort complexe car, en plus d'avoir à schématiser notre code afin de le séparer sur plusieurs ordinateurs sans perdre l'intégrité du code original, il faut gérer adéquatement les transmissions et la gestion supplémentaire de données

entre ordinateurs. D'où la nécessité de développer, d'une manière savante, une méthode capable de partager efficacement les tâches entre les ordinateurs.

Afin de schématiser le code, il faut noter que la plupart des simulations concernées par notre projet sont réalisées par un logiciel à interface graphique basé sur un langage de programmation par "diagramme-bloc" tel que Matlab/Simulink. C'est un environnement de simulation de systèmes très avancé où Matlab est l'outil numérique de calcul tandis que Simulink fournit des capacités de simulation ainsi que des schémas fonctionnels qui facilitent le développement des modèles. Cet environnement nous facilite la tâche car il offre déjà une représentation graphique du code. Cette représentation schématique n'est toutefois pas celle à laquelle nous sommes habitués. Avant de développer des algorithmes d'ordonnancement pour affecter les parties du code sur plusieurs ordinateurs, il faut développer une méthode de génération du graphe de précédence des tâches.

Pour effectuer parallèlement le calcul, nous utiliserons le logiciel Rt-lab, un environnement développé par *Opal-rt Technologies Inc* pour permettre le traitement parallèle des simulations. Il fournit en outre tout le matériel, logiciels et applications relatifs à cette technologie. Cet environnement permet actuellement d'avoir un code parallèle à partir du schéma fonctionnel dessiné sous un logiciel graphique tel Simulink. Mais, jusqu'à maintenant, la parallélisation se fait manuellement car Rt-lab ne permet pas de décider de l'affectation des tâches aux différentes unités de calcul. C'est un ingénieur qui intervient dans cette phase pour faire le travail.

L'objet de recherche de ce mémoire est de développer une méthode automatique de parallélisation dans l'environnement Matlab/Smulink/Rt-lab. Ce projet comporte deux phases : la phase de la génération du graphe des tâches et celle de la séparation et d'équilibrage du modèle. Dans la première nous travaillerons dans les différentes "couches" technologiques afin de générer le graphe de précédences des tâches soit à

partir du modèle Simulink soit à partir du programme écrit en Matlab. Cette tâche est technologiquement difficile, mais grâce à l'expertise de Opal-rt à naviguer dans les différentes structures de Matlab, nous allons relever ce défi. Pour l'autre phase, soit l'affectation des tâches, il s'agit d'un problème très difficile et relativement connu du domaine de la recherche opérationnelle, plus particulièrement de celui de l'ordonnancement. Nous allons toutefois adapter les heuristiques aux particularités de notre problème et explorer les bénéfices d'utiliser la duplication des tâches ainsi que les métaheuristiques.

Ainsi ce mémoire est constitué de cinq chapitres. Le premier chapitre dresse la définition et la mise en contexte du problème pratique. Par la suite nous présentons une revue de la littérature au chapitre 2. Le chapitre 3 est consacré au problème de génération du graphe des tâches. Le chapitre 4 traite de la formulation mathématique du problème d'ordonnancement et présente la résolution du problème par des heuristiques. Finalement, le chapitre 5 est consacré aux tests et aux résultats.

CHAPITRE 1

PRÉSENTATION ET MISE EN CONTEXTE DU PROBLÈME

L'objectif du présent chapitre est la mise en contexte du projet afin de bien connaître l'environnement d'étude. Nous commençons par une présentation de la compagnie Opal-rt, avec qui le projet est réalisé. Ensuite, nous présentons les structures générales des logiciels de *programmation graphique* de l'ensemble Matlab/Simulink. Ce qui nous permettra également de présenter le problème de *génération du graphe des tâches.* Après, nous dresserons une présentation générale de l'environnement Rt-lab. Finalement, dans la dernière partie de ce chapitre, on exposera la procédure actuelle de séparation manuelle du modèle de simulation suivie d'une définition de la problématique de *séparation et d'équilibrage des tâches.*

Opal-rt Technologies Inc. est une jeune compagnie technologique fondée en 1997 par des ingénieurs spécialistes en développement de systèmes en temps réel. Elle est composée d'un groupe de scientifiques, provenant de différents domaines : génie électrique, génie informatique, génie mécanique et aérospatiale. C'est donc une équipe dotée d'une grande expérience dans le domaine de design de commandes et de la simulation avec les logiciels Matlab et Matrixx.

Opal-rt travaille en collaboration avec plusieurs compagnies spécialisées dans les domaines des systèmes de contrôle et des outils de conception : The Mathworks Inc., QSSL(QNX) Inc., Sederta Inc., Wind River Inc. Parmis ses clients on trouve l'Agence Spatiale Canadienne (ASC), Ford, Bombardier, Embraer, Pratt & Whitney, Gué et General Motors. La technologie qu'elle développe est exportée à travers le monde. Ses distributeurs internationaux couvrent actuellement plusieurs pays

tels que : l'Allemagne, la Chine, l'Autriche, la Corée, la France, le Japon, la Suisse et les États-Unis.

Opal-RT offre une gamme complète de services d'ingénierie et de génie-conseil, y compris la mise à disposition et l'intégration de matériels informatiques, le développement de logiciels, la simulation et la modélisation, la résolution des problèmes et la formation. Tous ces services tournent principalement autour de son principal domaine d'expertise, la simulation en temps réel distribuée. En outre, elle développe plusieurs boîtes–à–outils pour le logiciel Matlab, particulièrement des outils pour aider et faciliter le développement d'applications sous Simulink.

La ligne des produits Opal-rt est spécialement conçue pour les développeurs et concepteurs de systèmes de contrôle, particulièrement dans les domaines de l'aéronautique, de l'aérospatiale, de "l'automatisation", de la robotique et des systèmes automatisés industriels. Son principal produit, qui forme le noyau de sa technologie, est Rt-lab. C'est un environnement qui permet le traitement parallèle de modèles Simulink (Matlab) et SystemBuild (Matrixx) par le biais d'une technologie flexible et accessible. Il rend possible la réalisation et le contrôle de simulations distribuées sur des ordinateurs PC réguliers, car Opal-rt n'utilise pas des super-ordinateurs mais plutôt des équipements "standards". Rt-lab et Matlab/Simulink forment une excellente plate-forme pour la création et le design de systèmes de contrôle de simulation.

1.1 L'environnement informatique de Rt-lab

L'environnement informatique avec lequel Rt-lab fonctionne est relativement complexe. Il est donc essentiel, afin de comprendre nos travaux sur la génération des graphes, de connaître ses principales composantes. Dans ce but, nous présentons tout d'abord un aperçu de la conception graphique. Ensuite, une présentation de

Matlab et de Simulink permettra de comprendre comment fonctionnent ces logiciels. Finalement, nous présentons le fonctionnement de Rt-lab en détails.

1.1.1 Conception graphique

Le domaine de la simulation des systèmes dynamiques et celui de la simulation en général a connu d'énormes progrès dans la dernière décennie tant sur le plan matériel que conceptuel par l'arrivée de nouvelles plates-formes plus puissantes et de nouveaux modes de programmation et logiciels. Ainsi, les avancements au niveau du matériel permettent, maintenant, la simulation de systèmes beaucoup plus grands et plus complexes. Puis, au niveau conceptuel, une nouvelle génération de logiciel de simulation avec des interfaces graphiques beaucoup plus conviviales rendent la simulation plus facile et généralisable à plusieurs systèmes.

Dans cette catégorie, l'un des logiciels de simulation le plus répandu est Matlab/Simulink. Matrixx/SystemBuilt est son principal concurrent dans cette catégorie. Ce type de logiciels propose une excellente interface graphique pour faire la conception. Ainsi, l'ingénieur développe des blocs de programme sous forme de schémas fonctionnels: ce qui est beaucoup plus proche de la réalité qu'un langage de programmation et plus pratique pour la conception de systèmes de commande. Cependant, cette méthode a des limitations quand il s'agit de modéliser des systèmes complexes où les blocs du schéma fonctionnel sont difficilement identifiables aux composants du système réel. La limitation fondamentale réside en fait dans les habilités du concepteur à s'assurer que le schéma fonctionnel inclut l'information explicite de la causalité dans le système.

La conception graphique sur Matlab/Simulink permet donc de concevoir plus rapidement des systèmes de contrôle. Avant d'examiner Simulink plus en détails, nous

donnons un aperçu de Matlab.

1.1.2 Matlab

Matlab a été initialement développé pour faciliter l'accès au logiciel de matrices développé dans le cadre des projets de Linpack et Eispack. Depuis, le logiciel s'est transformé en un système et un langage de programmation interactifs pour le calcul général et la visualisation scientifiques et techniques. Toutefois, l'élément de base de Matlab reste la *matrice*. Les commandes de Matlab sont exprimées sous une forme très semblable à celles utilisées dans les domaines des mathématiques et de l'ingénierie. Par exemple, $B = AX$, où A, B, et X sont des matrices, est écrit simplement : $B = A * X$ dans Matlab. Et pour la division de A par B, on écrit $X = A/B$. Il n'y a donc aucun besoin de programmer explicitement des exécutions de matrice. De même, la transposition ou l'inversion d'une matrice ne nécessitent que l'utilisation d'opérateurs.

L'encodage des problèmes dans Matlab est donc généralement beaucoup plus rapide que dans un langage de haut niveau tel que C ou Fortran. Il y a des centaines de fonctions intégrées incluses dans le Matlab de base et il y a les "boîtes-à-outils" facultatives pour des applications spécifiques telles que le traitement des signaux et l'optimisation. La plupart des fonctions dans Matlab et des boîtes-à-outils sont écrites dans le langage de Matlab, le Script M. Le code source des sous-routines est accessible pour être modifié au besoin.

Il y a deux versions de base du logiciel, la version professionnelle et la version étudiante "Student Edition". La version étudiante, distribuée par Prentice-Hall, est seulement limitée dans la taille maximale des matrices. La version professionnelle, distribuée par le MathWorks, inc., a simplement une puissance de calcul accrue par

rapport à la version étudiante.

1.1.3 Simulink

Simulink est un logiciel interactif de modélisation, simulation et analyse de systèmes dynamiques. C'est un logiciel avec une interface graphique qui permet à des systèmes d'être modélisés en construisant un schéma fonctionnel sur l'écran. Il peut manipuler les systèmes linéaires, non linéaires, en temps continu, en temps discret, hybride, multivariables, "monorate" et "multirate".

Simulink fonctionne sur des postes de travail Unix (ou des PC avec Linux) avec X-Windows, sur des PC avec Microsoft Windows, et sur les Macintosh. Il profite pleinement de la technologie de fenêtrage et de l'interaction graphique pour ses interactions avec le concepteur. Simulink est entièrement intégré à Matlab. En fait, Simulink a besoin de Matlab car un programme Simulink est par la suite traduit dans le langage de Matlab avant d'être exécuté. Inversement, Matlab peut contrôler les composantes de Simulink via des programmes écrits en Script M.

Pour la modélisation, Simulink utilise une interface graphique GUI (Graphical User Interface) qui permet, à l'aide de blocs prédéfinis et de connecteurs, de créer un modèle rapidement sans avoir à écrire de code. Un modèle est en général constitué de blocs et de lignes qui les relient (voir figure 1.1). Dans le contexte de modélisation de contrôles simulés, les blocs et les lignes proviennent de la boîte à outils "Control System Toolbox" où ils sont créés par l'utilisateur en respectant les règles de cette boîte à outils. La conception est hiérarchisée car un bloc (ou un sous-système) est par la suite éclaté pour le définir plus en détail (voir figure 1.2). En outre, Simulink gère automatiquement la simulation en relançant le modèle n fois. L'exécution simple du modèle est appelée un "pas d'intégration".

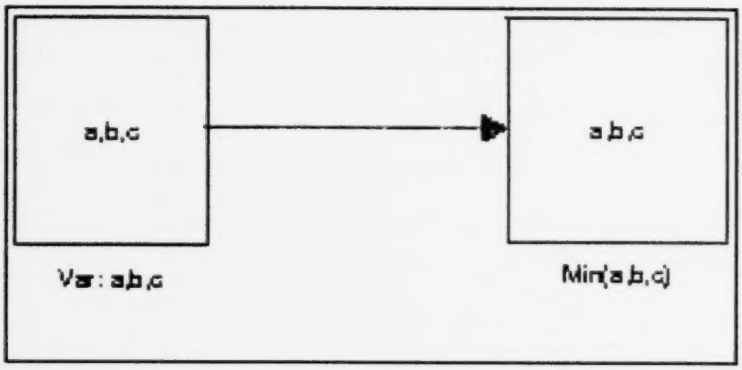

Figure 1.1 : Exemple d'un modèle sous Simulink (calcul du minimum de trois nombres)

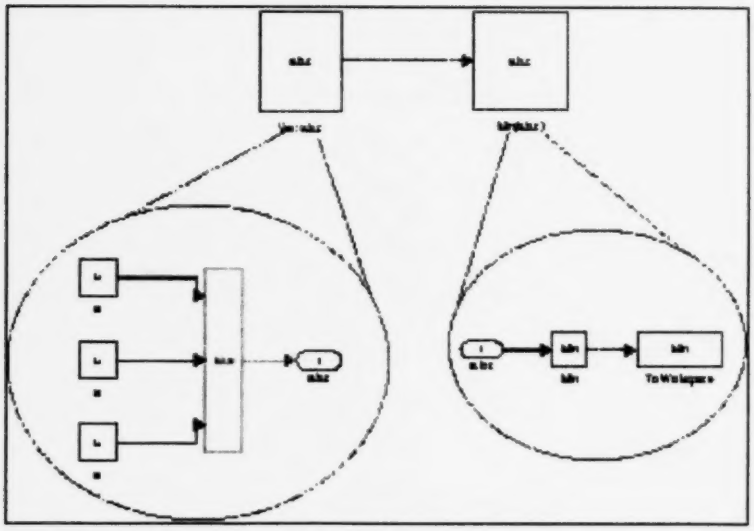

Figure 1.2 : Le modèle de la figure 1.1 éclaté

1.1.4 Présentation du logiciel Rt-lab

L'architecture du logiciel Rt-lab est présentée à la figure 1.3. Rt-lab permet de distribuer tout modèle (de contrôle) écrit en langage Simulink (ou SystemBuild) de façon semi-automatique sur plusieurs CPUs de type Intel reliés entre eux avec le système d'exploitation RT-OS QNX. La distribution n'est que semi-automatique car l'utilisateur doit identifier, dans Simulink, quel bloc sera exécuté sur quel CPU. Une fois cette attribution réalisée sous Simulink, Rt-lab s'occupe de la séparation et de la génération du modèle en langage C en autant de processus indépendants que de CPUs considérés. La génération en code C utilise le compilateur de RealTime-Workshop (RTW), un module de Matlab (voir figure 1.4). Cependant d'autres commandes sont ajoutées par Rt-lab au programme en C pour permettre l'exécution dans un système distribué.

Deux CPUs installés sur le même PC utiliseront la mémoire partagée pour l'échange des données tandis que la communication entre deux CPUs installés sur deux ordinateurs différents se fera via Firewire (400 Mbits/sec) ou Giganet (1,2 Gbits/sec). Rt-lab inclut certains outils de monitoring qui permettent, par exemple, de mesurer le temps d'exécution par CPU ou par pas d'intégration. Par ailleurs, en vue de la compilation sous RTW, tout code C écrit antérieurement à la main peut être intégré dans un schéma Simulink par le biais des fonctions "C-S ", plutôt que réécrit à l'aide de blocs Simulink. Enfin, la visualisation en temps réel des données mesurées peut se faire, si besoin est, à l'aide d'une interface autre que celle de Simulink, par exemple LabView ou Altia. Pour finir, le même logiciel Rt-lab permet toute simulation sous l'environnement Windows-NT dans le cadre seulement de simulations accélérées. Toutefois, les simulations en temps réel demandent absolument l'environnement QNX..

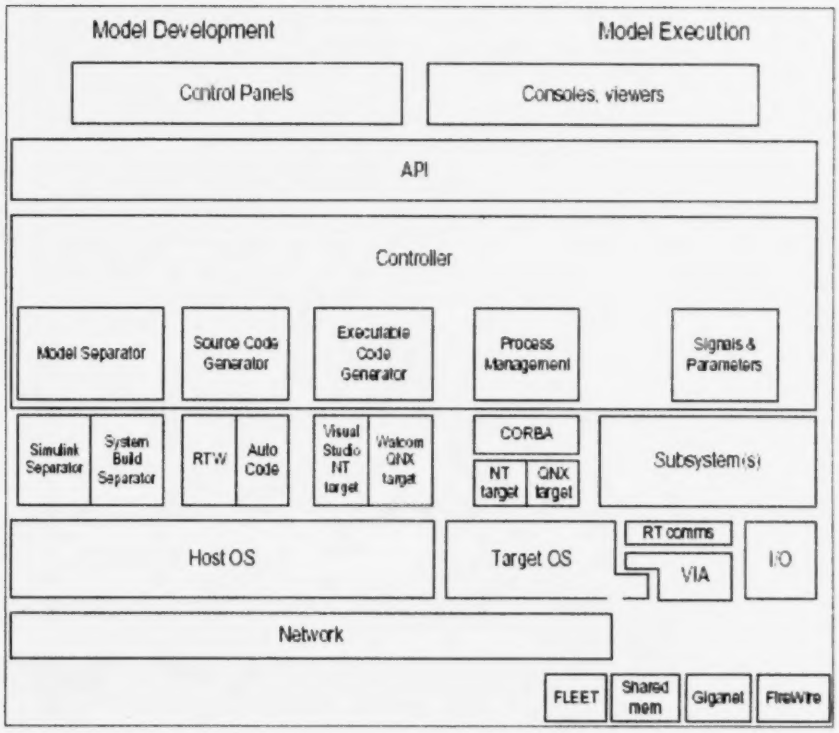

Figure 1.3 : L'Architecture de Rt-lab

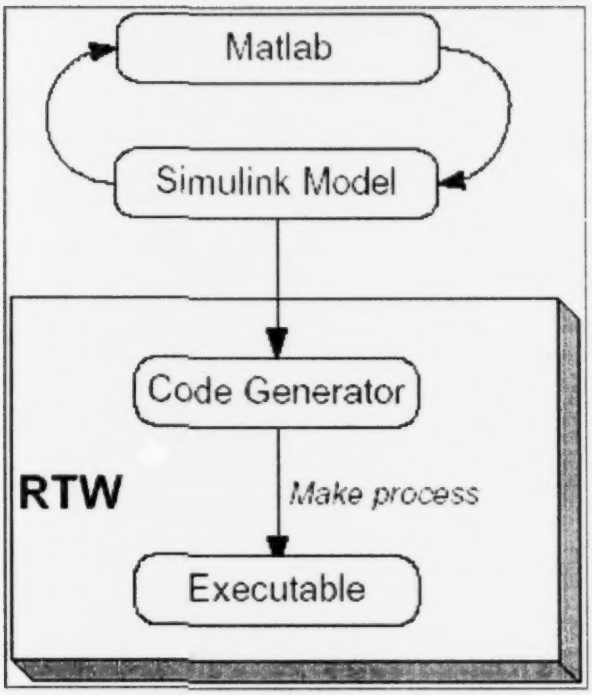

Figure 1.4 : Mécanisme de RTW(Real-Time Workshop)

1.1.5 Procédure de parallélisation sous Rt-lab

Comme expliqué précédemment, Rt-lab ne permet qu'une distribution semi-automatique. L'utilisateur doit ainsi préparer manuellement la séparation des tâches d'un modèle avant de passer le relais à Rt-lab pour générer un code parallèle. Cette procédure manuelle doit minimiser le temps total d'exécution du modèle. Pour ce faire, on doit équilibrer la charge de chaque CPU, en terme de temps de calcul, mais minimiser les temps de communication inter CPUs qui sont très coûteux (les tâches requerrant les mêmes variables impliquent une communication si elles sont exécutées sur différents CPU). Présentement, la séparation et l'équilibrage du modèle se font selon une expertise développée par la pratique. Cette procédure manuelle est présentée schématiquement à la figure 1.5. Elle se résume en les trois étapes suivantes :

- Étape 1 : Tout d'abord on détermine le nombre de CPUs nécessaires pour que l'exécution du modèle soit faisable dans le pas de temps imposé. Pour cela, on fait exécuter le modèle sur un seul processeur afin d'obtenir une estimation de la durée totale du calcul. Puis on divise cette durée par le pas d'intégration. Ce dernier est imposé dans le cas d'un temps réel et estimé dans le cas d'une simulation avancée, pour l'obtention d'un niveau de précision requis pour le modèle.

- Étape 2 : Disposant du nombre de CPUs, des caractéristiques des tâches et des liens de communication, on procède à la séparation du modèle. On regroupe les blocs et les sous–systèmes par processeur de façon à créer des ensembles équilibrés et avec le minimum de liens de communication. On crée le groupe CPU-Master et les groupes CPU-Slaves 1, 2 et ... m (leurs noms doivent être obligatoirement précédés du préfixe "SM_" pour le Master et du préfixe "SS_" pour les Slaves). Ainsi Rt-lab reconnaît les groupes et pourra créer les codes parallèles de chaque CPU (le "Master" et les "Slaves"). Avant

de regrouper, il faut rapprocher graphiquement les blocs à inclure dans le même groupe avant la création des sous-systèmes, sinon Simulink envoie un message d'erreur.

- Étape 3 : Une fois que le modèle graphique est séparé et que la configuration de l'ensemble est arrêtée, on lance Rt-lab pour la génération et la compilation du modèle parallèle. On vérifie d'abord que le temps d'exécution total sur chaque CPU est inférieur au pas d'intégration requis. Puis on regarde tous les temps de communication utilisés. On refait les étapes 2 et 3 jusqu'à l'obtention d'un temps total de calcul le plus correct possible.

1.1.6 Le langage bloc de Simulink

Avant de passer à l'automatisation de la parallélisation, il faut en savoir un peu plus sur l'infrastructure d'un modèle Simulink/Matlab/Rt-lab. Les fichiers générés après séparation du modèle par Rt-lab sont nombreux. Rt-lab génère pour chaque groupe - à exécuter sur un CPU - un fichier ".mdl" (modèles sous Simulink), un fichier ".c" (programme en C), un exécutable ".exe" et quelques autres programmes. La figure 1.6 montre les fichiers générés pour chaque CPU. Ainsi, un modèle séparé sur deux CPUs, l'un appelé Slave et l'autre Master, possédera chacun de ces fichiers en double.

Le principal fichier est celui possédant l'extension ".mdl". Pour chaque modèle conçu graphiquement, Simulink le traduit dans un langage texte. Les données du modèle sont ainsi enregistrées dans un fichier ".mdl", un fichier ASCII. La structure de ce dernier permet d'obtenir la composition du modèle d'une façon structurée. La figure 1.7 montre un exemple de fichier ".mdl" ouvert par un éditeur de texte. On peut voir la structure hiérarchique du modèle : les systèmes et sous systèmes, les

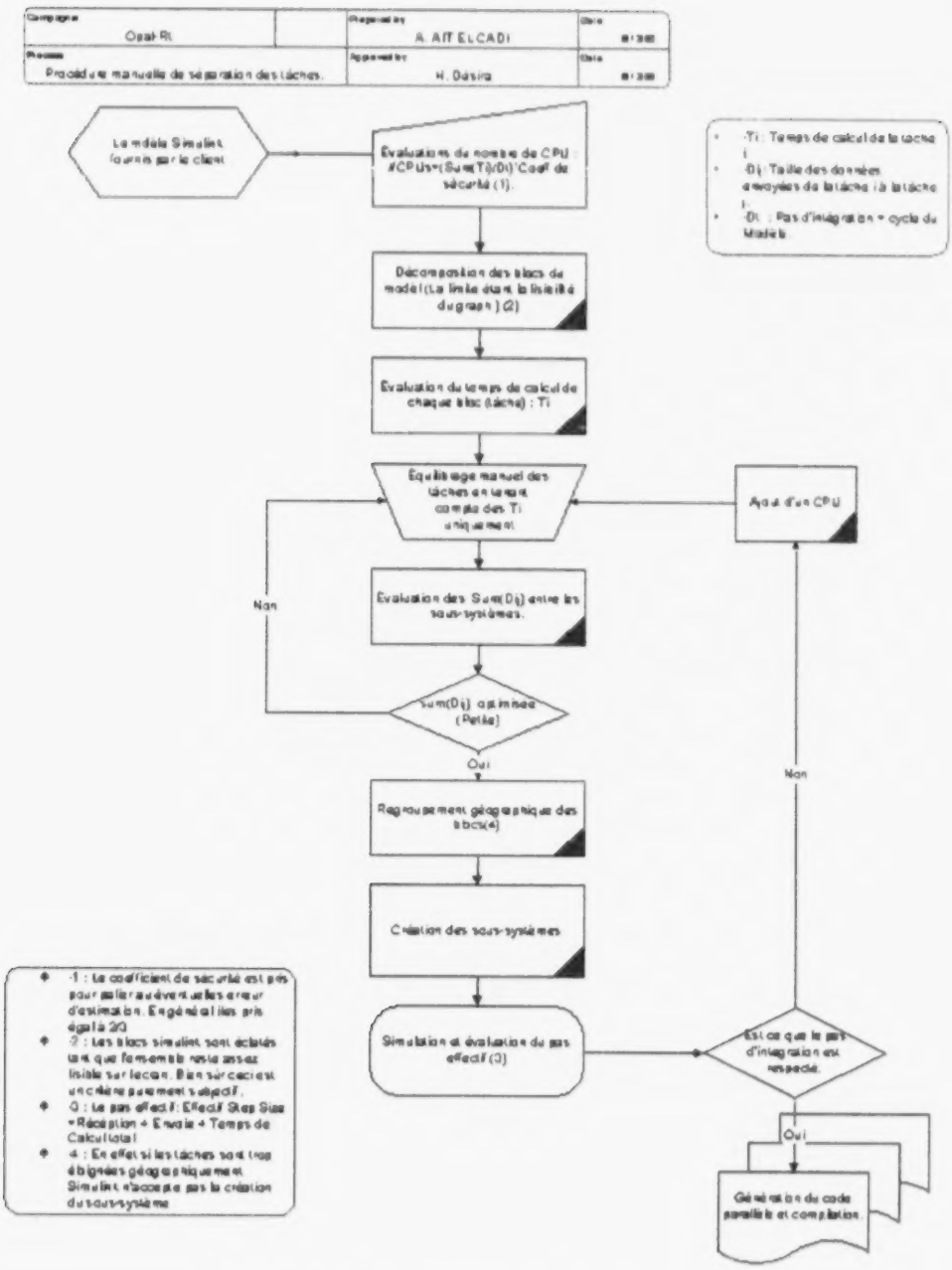

Figure 1.5 : Procédure manuelle de séparation

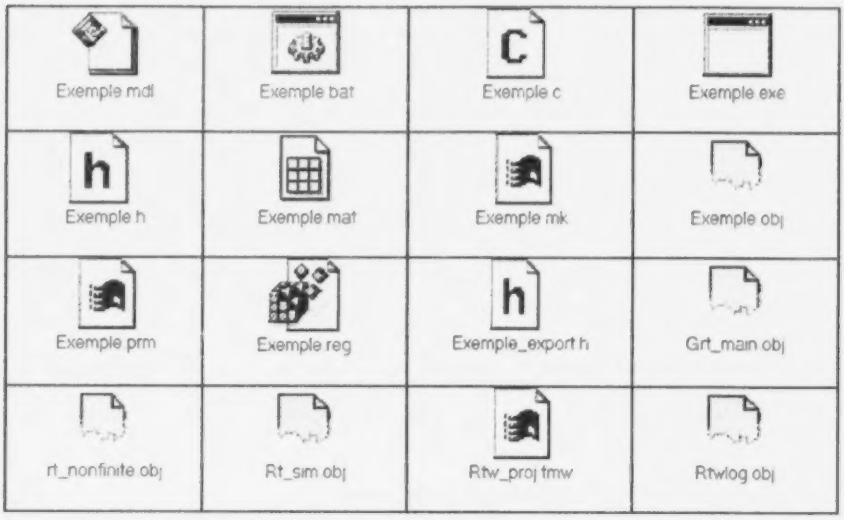

Figure 1.6 : Ensemble des fichiers générés par Rt-lab pour le modèle "exemple.mdl"

tâches et les liens entre les tâches. Ce fichier est fort intéressant pour notre projet de génération automatique du graphe des tâches car on pourrait agir directement sur ce fichier texte pour obtenir des informations pertinentes sur la structure du code d'exécution du modèle (nous reviendrons en détails sur ces fichiers dans les prochaines sections).

Le fichier ".c", généré par RTW et associé à chaque modèle, n'est pas facilement exploitable. La figure 1.8 montre la structure du code généré. Ce fichier n'est pas facilement transportable. Il ne peut être facilement exécuté sur une autre plate-forme car utilise plusieurs librairies spécifiques à Simulink (#include "simstruc.h", #include "rtwlog.h", etc). Ce fichier est généralement composé de 4 sections :

- Start modèle : Il est exécuté une seule fois au début de la simulation. Il sert à initialiser les états.

- Outputs : Il calcule les sorties de chaque bloc à chaque étape de la simulation.

```
/* [BOF] Exemple.mdl */
Model {
      Name      ``Exemple''
      Parametres du modeles (Pas dintégration, durée de simulation)...
      System {
            Name        ``Exemple''
            Location    [2, 70, 1022, 720]...
            Block {
                  BlockType   SubSystem
                  Name        ``Min(a,b,c)''...
                  System {
                        Name        ``Min(a,b,c)''
                        Location    [129, 170, 348, 236]...
                        Block {
                              BlockType   Inport
                              Name        ``a,b,c''...}
                        Block {
                              BlockType   MinMax
                              Name        ``Min''...}
                        Block {
                              BlockType   ToWorkspace
                              Name        ``To Workspace''...}
                        Line  {
                              SrcBlock    ``Min'';          SrcPort   1
                              DstBlock    ``To Workspace''; DstPort   1}
                        Line  {
                              SrcBlock    ``a,b,c'';        SrcPort   1
                              DstBlock    ``Min'';          DstPort   1}}}
                  Block {
                        BlockType   SubSystem
                        Name        ``Var: a, b, c''...
                        System {
                              Name        ``Var: a, b, c''
                              Location  [24, 116, 243, 290]...
                              Block {
                                    BlockType   Mux
                                    Name        ``Mux''...}
                              Block {
                                    BlockType   Constant
                                    Name        ``a''...}
                              Block {
                                    BlockType   Constant
                                    Name        ``b''...}
                              Block {
                                    BlockType   Constant
                                    Name        ``c''...}
                              Block {
                                    BlockType   Outport
                                    Name        ``a,b,c''...}
                              Line  {
                                    SrcBlock    ``a'';      SrcPort   1
                                    DstBlock    ``Mux'';    DstPort   1}
                              Line  {
                                    SrcBlock    ``b'';      SrcPort   1
                                    DstBlock    ``Mux'';    DstPort   2}
                              Line  {
                                    SrcBlock    ``c'';      SrcPort   1
                                    DstBlock    ``Mux'';    DstPort   3}
                              Line  {
                                    SrcBlock    ``Mux'';    SrcPort   1
                                    DstBlock    ``a,b,c'';  DstPort   1}}}
                  Line  {
                        SrcBlock    ``Var: a, b, c'';   SrcPort   1
                        DstBlock    ``Min(a,b,c)'';     DstPort     1}}}
/* [EOF] Exemple.mdl */
```

Figure 1.7 : Le fichier ".mdl" du moèle de la figure 1.1

```
/* [BOF] Exemple.c */
 * Exemple.c
* Real-Time Workshop code generation for Simulink model ``Exemple.mdl''.
* Model Version                        : 1.7
 * RTW file version                    : 3.0 $Date: 1999/09/30 23:33:29$
 * RTW file generated on               : Mon Jun 26 16:25:12 2000
 * TLC version                         : 1.0 (Jan 15 1999)
 * C source code generated on          : Mon Jun 26 16:25:12 2000
* Relevant TLC Options :
 *          InlineParameters                    = 0
 *          RollThreshold                       = 5
 *          CodeFormat                          = RealTime
* Simulink model settings:
 *          Solver                    : FixedStep
 *          StartTime                 : 0.0 s
 *          StopTime                  : 10.0 s
 *          FixedStep                 : 1.0 s
 */
# include <math.h>
# include <string.h>
# include ``Exemple.h''
# include ``Exemple.prm''

/* Start the model */
void MdlStart(void) {                    /* start code */
        /* ToWorkspace Block: <S1>/To Workspace */
        rtPWork.s1_To_Workspace.LoggedData = rt_CreateLogVar(rtS,'Min',
                                             SS_DOUBLE0,0,0,1,1,1,1,1);
        if (rtPWork.s1_To_Workspace.LoggedData == NULL) return;
}
/* Compute block outputs */
void MdlOutputs(int_T tid) {                /* local block i/o
variables */
        real_T rtb_s2_a;
        real_T rtb_s2_b;
        real_T rtb_s2_c;
        real_T rtb_s1_Min;
/* Constant Block: <S2>/a */
        rtb_s2_a = rtP.s2_a_Value;
        /* Constant Block: <S2>/b */
        rtb_s2_b = rtP.s2_b_Value;
/* Constant Block: <S2>/c */
        rtb_s2_c = rtP.s2_c_Value;
/* MinMax Block: <S1>/Min */
        {
              real_T min = rtb_s2_a;
              if (rtb_s2_b < min) { min = rtb_s2_b; }
              if (rtb_s2_c < min) { min = rtb_s2_c; }
              rtb_s1_Min = min;
        }
/* ToWorkspace Block: <S1>/To Workspace */
      t_UpdateLogVar((LogVar*)(rtPWork.s1_To_Workspace.LoggedData),rtb_s1_Min);
}
/* Perform model update */
void MdlUpdate(int_T tid) {              /* (no update code required) */
}
/* Terminate function */
void MdlTerminate(void) {                 /* (no terminate code required) */
}
# include ``Exemple.reg''
/*[EOF] Exemple.c */
```

Figure 1.8 : Le fichier ".c" représentant le code généré par RTW pour le modèle Simulink de la figure 1.7 (et 1.1)

- Update : Il met à jour les états du système à la fin de chaque étape.
- Terminate : Il est exécuté à la fin de la simulation.

Cependant, l'information structurée du modèle Simulink -tel qu'il est- est plus intéressant que les fichier ".mdl" et ".c". En effet, Matlab dispose d'un descripteur qui rend le modèle Simulink très transparent. Ce descripteur n'est rien d'autre qu'un ensemble de fonctions (*comme get_param( ), set_param( ), etc*) qui scrutent le modèle et rendent ces paramètres visibles et contrôlables de l'extérieur du modèle.

En plus, l'integration de Simulink dans matlab permet de contrôler le modèle à partir du shell Matlab. Un script M peut accéder facilement à toutes les caractéristiques du modèle, ce qui permet une lecture automatique du modèle via un code et, ainsi, facilite la génération du graphe des tâches. De plus, nous pouvons agir sur les paramètres du modèle, toujours à l'aide d'un script M, pour modifier la constitution de notre modèle, ses composantes et leurs emplacements. C'est la solution technique que nous avons retenu afin de générer automatiquement le modèle séparé.

1.2 La problèmatique

La structure de Simulink et l'environnement Rt-lab/Matlab/Simulink, décrits ci-dessus, présentent des opportunités d'accélération de la procédure manuelle de parallélisation. Cette dernière procédure est fastidieuse, longue, consomme beaucoup de temps et est propice à des erreurs de conception. Ainsi, dans la suite de ce mémoire, nous établirons une procédure automatique permettant d'accélérer la tâche de parallélisation d'un système distribué en plus d'augmenter la qualité de la solution ou du moins assurer une constance de qualité.

L'automatisation de cette procédure nécessite deux grandes phases :

- Définition des tâches (niveau de décomposition, contraintes de précédences, de hiérarchie, de communication et d'accès aux ressources) dont le résultat est la génération d'un graphe de tâches.
- Séparation et équilibrage (équilibrage de ces dernières tâches en tenant compte des contraintes associées)

La première phase consiste en la lecture et la synthèse automatique du modèle Simulink en un graphe de tâches plus facile à exploiter et à traiter. Puisque c'est une phase de pré-traitement des données d'un modèle Simulink, il s'agit donc de la conception d'un "parser" (analyseur syntaxique) dans Simulink. Ainsi, la génération du graphe acyclique des tâches demande de naviguer à travers tous les blocs d'un modèle Simulink afin de leur apposer une nouvelle codification pour un traitement ultérieur. Cette partie est traitée dans le chapitre 3.

La deuxième phase consiste en un problème d'ordonnancement. Il s'agit de la séparation d'un ensemble de n tâches sur m processeurs en tenant compte des contraintes de communication, de précédence et de hiérarchie. Ceci est traité dans le chapitre 4.

CHAPITRE 2

REVUE DE LA LITTÉRATURE

Pour résoudre notre problématique - définie au chapitre 1 - nous avons investigué la littérature relative aux différents domaines liés à notre problème, soit le calcul parallèle, un problème spécial de la grande classe de problèmes d'ordonnancement. Ainsi nous avons fait un survol de quelques travaux reliés aux problèmes génériques suivants : (i) **Calcul parallèle** (Darte et al., 2000; Stone, 2000; Chen et Lin, 2000); (ii) **Les simulations distribuées** (Chandy et MISRA, 1979; Dado et al., 1993); (iii) **Ordonnancement** (French, 1982; Carlier, 1989); (iv) **Équilibrage des lignes d'assemblage** (Urban, 1998; Muth, 1963); (v) **Gestion de projet** (Milon, 1999; Sprecher, 2000)

Le problème d'ordonnancement est un sujet qui intéresse aussi bien les informaticiens que la communauté de la recherche opérationnelle. Ces deux communautés ont initialement oeuvré étroitement sur cette thématique. Cependant, depuis les années 80, les deux communautés travaillent trop souvent séparément. Ainsi, les informaticiens s'attardent plus à bien décrire les problèmes pratiques de calcul parallèle mais s'en tiennent au minimum à formaliser les problèmes et à approfondir les méthodes de résolution. D'un autre côté, la communauté de la recherche opérationnelle publie de bons articles sur les problèmes trop simplifiés des besoins réels de la pratique. Faire une revue exhaustive de la littérature sur le calcul parallèle est une tâche ardue à cause des liens très peu étroits entre ces deux communautés scientifiques et à cause de la quantité de travaux réalisés dans ce domaine. Nous nous contentons de présenter quelques travaux et idées de ce grand domaine.

Avant d'attaquer le problème de parallélisation en tant que tel, nous présentons tout d'abord les différentes approches de parallélisation d'un programme. Par la suite, nous couvrons la génération du graphe des tâches, surtout dans le contexte de l'ensemble Simulink/Matlab, RTW et Rt-lab. Pour notre problème, nous présentons en premier lieu les travaux sur le formalisme mathématique. Ensuite, nous décrivons les différentes méthodes de résolution du problème de calcul parallèle et d'ordonnancement.

2.1 Les différentes approches de parallélisation

Pour paralléliser un programme, il existe différentes approches conceptuelles. Rault (1998) a recensé trois approches de parallélisation pour le cas de systèmes à équations différentielles. Elles sont les suivantes :

(1) **Parallélisation à "travers le problème"** : Il s'agit d'une approche qui s'attaque au phénomène physique lui-même pour en déduire une forme de parallélisation. Dans le cas du calcul d'orbite, il a parallélisé en séparant le calcul de l'effet de chacune des forces - agissantes sur l'orbite - sur différents CPUs. Quelques références dans ce domaine sont Beaugendre (1995) et Gear (1986).

(2) **Parallélisation à "travers le schéma"** : Cette approche s'attaque - dans le cas des équations différentielles - aux méthodes d'intégration. En quelque sorte, elle traite le schéma modélisant le phénomène physique. Un exemple serait de paralléliser le produit des matrices. On y trouve comme travaux ceux de Burrage (1995), Butcher (1993a et 1993b) et Chartier (1994).

(3) **Parallélisation à "travers le temps"** : Cette approche est basée sur les travaux de Bellen et Zennaro (1989) et ceux de Chartier et Philippe (1993).

Cette méthode consiste à paralléliser le problème d'intégration sur des sous-intervalles I_i pour chacun des processeur i au lieu de travailler sur l'intervalle global $I = \cup I_i$. Donc, les processeurs font les mêmes calculs sur des intervalles différents. Pour cette dernière, la difficulté réside dans la détermination des conditions initiales pour chaque calcul sur chacun des sous-intervalles.

Cogné (1997) présente une autre classification des différentes approches de parallélisation. Son travail s'applique à l'étude de robots et de systèmes dynamiques. Il propose la classification suivante :

(1) **Parallélisation par le formalisme**. Une approche qui s'attaque au phénomène à la base pour créer un modèle - physique et/ou mathématique - parallèle.

(2) **Parallélisation par la topologie**. On considère la composition géométrique - topologique - du robot ou du système. On s'en inspire pour concevoir un modèle parallèle. Exemple, pour simuler un robot à deux bras, on fera en sorte que chaque processeur fasse le calcul pour un bras.

(3) **Parallélisation par le code**. Cette approche s'attaque au code numérique d'une façon indépendante du phénomène.

À partir de ces deux visions de classification, on voit qu'on a surtout des approches qui s'attaquent à la parallélisation du phénomène physique soit d'un point de vue topologique soit d'un point de vue de séparation des causes du phénomène. Dans le contexte de Rt-lab, les problèmes sont déjà modélisés sous Simulink : nous n'avons pas cette option mais nous pouvons formuler des recommandations aux concepteurs des modèles. L'autre grande approche suggère d'attaquer la parallélisation au niveau du code faisant la résolution du modèle. Cette dernière est plus difficile que via le modèle car le code est moins riche en information sur la structure du problème à résoudre que ne l'est le modèle.

Notre approche est bâtie entre ces deux grandes approches. Certes, nous ne pouvons traiter le phénomène modélisé, mais Simulink - un langage de schématisation fonctionnel - donne une certaine représentation quasi-topologique de ce dernier, sans toutefois le dévoiler en tant que tel. Nous avons le code "C" pour résoudre le modèle - qui offre la parallélisation par le code - mais ces fichiers sont plus difficiles à traiter que le modèle Simulink, selon notre point de vue.

Nous avons donc à choisir entre la parallélisation du code numérique et le modèle fonctionnel car, dans Matlab, nous avons la résolution du modèle écrit avec le langage"C" généré par RTW de Matlab, et le modèle fonctionnel dessiné sous Simulink. Après étude du contexte Simulink et ses modèles (Mathworks, 2000c), de la plate forme de Matlab (Mathworks, 2000a), du code "C" généré par RTW (Mathworks, 2000b) et de l'environnement Rt-lab (Opal-rt, 2001), nous avons choisi le modèle Simulink pour paralléliser notre problème. En effet, le code "C" n'est pas très bien exploitable et au niveau de ce dernier on perd certaines informations topologiques utiles pour la parallélisation. D'un autre côté, le modèle Simulink offre une représentation graphique assez similaire à celle dont nous sommes habitués à traiter en recherche opérationnelle, soit le graphe acyclique orienté des tâches (Directed Acyclique Graphe (DAG)).

2.2 La génération du graphe des tâches

Il y a très peu de travaux sur la génération du DAG. Tous les travaux en ordonnancement et en calcul parallèle assument qu'on a un DAG dès le départ. Il y a quelques publications sur la génération du DAG à partir d'un programme dans un langage de type "C" (Solar et Inostroza, 2002) mais ceci est très loin de notre réalité car les boucles (while ... do) ne sont pas présents dans nos modèles de plus haut niveau.

Un aspect intéressant dans la génération des graphes est le degré de granularité, soit la taille de la tâche la plus petite dans un graphe. Gerasoulis et Yang (1993) ont montré que le degré de granularité doit être pris en considération dans l'élaboration d'une stratégie de parallélisation d'un graphe. C'est que le concept de la granularité est cruciale dans la parallélisation : elle permet de savoir si le mode parallèle est justifié par rapport au mode séquentiel, selon la taille des tâches à exécuter et les temps de communication entre les processeurs. En effet, Stone (1987) a démontré l'existence d'un seuil de granularité au dela duquel la parallélisation n'est plus intéressante.

2.3 Formulation mathématique du problème

Selon notre revue de la littérature, peu de travaux dressent une formulation mathématique complète [1] du problème de paralléliser l'exécution d'un DAG. Généralement la plupart des articles attaquent directement la résolution du problème via des heuristiques sans présenter de modèle formel. D'autres donnent des modèles ne tenant compte que d'une partie des contraintes : Urban (1998) et Darte et al. (2000) ont établi des modèles sans contraintes de communication. Chen et Lin (2000) ont présenté un modèle tenant compte de la contrainte de communication mais avec des contraintes de capacités - capacité maximale de calcul par CPU.

Cependant, nous constatons qu'il existe principalement deux approches de modélisation mathématique. La première vise à déterminer l'horaire des tâches en mettant l'emphase sur les décisions du temps de début d'exécution des tâches (Urban, 1998). La deuxième s'intéresse à la séquence des tâches, ce qui simplifie la complexité du modèle au niveau de sa résolution. La première formulation d'un modèle par

[1] Une modélisation tenant compte de toutes les contraintes de précédence et de communication.

séquence à notre problème de calcul parallèle est donné par Blazewicz et al. (1986). Elle est basée sur la caractérisation de la position d'exécution de chaque tâche sur son CPU d'affectation. Maculan et al. (1999) ont amélioré ce modèle mais n'ont pas réussi à le résoudre en tant que tel. Dans le chapitre 4, nous présentons cette formulation " par séquence" adaptée à notre problèmatique, en plus d'une formulation "horaire", qui est tout de même intéressante car elle est plus simple à comprendre.

2.4 Méthodes de résolution

Le problème d'odonnancement est NP-Complet (Murty, 1994b; Garey, 1979). Les méthodes exactes de résolutions sont aussi très complexes et nécessitent beaucoup de temps de calcul. Une classification des différents problèmes d'ordonnancement et leur complexité d'exécution a été proposée par Lawler et al. (1982) et Blazewicz et al. (1983) et complétée par Lawler et al. (1993) et Blazewicz et al. (1993). Les articles proposant des méthodes de résolution déterministes traitent seulement des problèmes simplifiés, du type temps de communication unitaire et plus petit que le temps d'exécution de toutes les tâches (voir par exemple Brucker (1995), Colin et Chrétienne (1991), Thersen (1998) et Vigo et Maniezzo (1997)). Nous sommes plutôt intéressés par les algorithmes pouvant traiter les problèmes généraux. Nous présentons, ici, seulement les différentes méthodes de résolutions pouvant attaquer notre problème d'origine.

2.4.1 Les heuristiques

Les heuristiques sont des approches de premier plan car elles sont les plus faciles à développer pour résoudre des problèmes complexes. Les principales heuristiques

sont de la famille "les heuristiques basées sur des règles de priorité" ou "Clustering".

2.4.1.1 Les heuristiques basées sur des règles de priorité

Les heuristiques basées sur des règles de priorité sont probablement les heuristiques les plus simples à développer. Ce sont des heuristiques de construction. Elles fabriquent des solutions d'une manière séquentielle, sans retour en arrière. L'approche est basée sur l'attribution d'une priorité à chaque tâche. La priorité est calculée selon un critère prédéfini. Puis les tâches disponibles sont affectées sur les CPUs selon leur priorité. Une tâche est dite "disponible" si tous ces prédécesseurs sont déjà affectés à un CPU. La qualité de la solution dépend du choix de la règle de priorité. Une règle donnée peut réaliser, pour le même type de problème, de bons résultats dans un contexte et de mauvais résultats dans un autre contexte. Conséquement, on utilise généralement plusieurs règles afin d'avoir plusieurs solutions pour le cas concerné, la meilleure étant retenue.

COMSOAL, développé par Arcus (1966), est la première et la plus populaire des heuristiques de règles de priorité appliquées à un problème d'ordonnancement. Elle a été créée pour un problème d'équilibrage de lignes d'assemblage. La priorité est basée sur la durée de la tâches (la tâche la plus courte en premier, ou la tâche la plus longue en premier) ou un choix aléatoire. Boctor (1995) propose un ensemble de règles hybrides basées sur la pondération de plusieurs règles de priorité. En général, dans les problèmes d'équilibrage de lignes d'assemblage on utilise des règles basées sur la taille des tâches. Dans les domaines de gestion de projet et de calcul parallèle, on utilise plutôt des règles plus raffinées comme celle donnée par le chemin critique, tel qu'expliqué en détail par El-Rewini et al. (1994). Saad (1995) et Maric et Jovanovic (1999) traitent également des heuristiques de règles de priorité pour les problèmes d'ordonnancement de tâches dans un environnement multiprocesseurs.

Une heuristique basée sur les règles de priorité est définitivement le premier algorithme à implanter dans un logiciel. Pour le calcul parallèle, la règle de priorité basée sur le chemin le plus long est la première règle à utiliser.

Lorsqu'on permet la duplication de certaines tâches sur plusieurs CPU, l'application d'une heuristique basée sur une règle de priorité augmente la quantité de calcul total mais on peut améliorer les solutions de parallélisation là où la durée de communication est très grande. C'est une bonne stratégie qui a été exploitée par Munier et Hanen (1997) et Colin et Chrétienne (1991), afin de trouver de meilleures solutions de parallélisation.

Les ingénieurs d'Opal-rt sont sceptiques quant aux bénéfices de la duplication des tâches. Donc nous nous limiterons à des heuristiques ne permettant pas la duplication des tâches. Des essais pourront toutefois être faits à même le modèle Simulink. À noter que la plateforme Rt-lab ne peut pas traiter d'une duplication de tâches qui ne serait pas explicitement sur le modèle Simulink.

2.4.1.2 "Clustering"

Le "Clustering" est une autre catégorie d'heuristiques utilisées pour la résolution de problèmes de parallélisation. Cette technique donne souvent de très bons résultats dans la parallélisation de systèmes distribués. Elle domine d'ailleurs les livres de calcul parallèle écrits par les informaticiens (voir Darte (2000), par exemple).

Le "Clustering" consiste à regrouper certaines tâches en un seul groupe - le "cluster" - qui sera affecté à un seul CPU. Une fois les "cluster" formés, ils sont ordonnancés en utilisant une heuristique basée sur la règle de priorité comme présenté par Sarkar (1989). Le "Clustering" est également utilisé pour résoudre d'autres problèmes typiques de la recherche opérationnelle, tel le problème de tournée de véhicules.

Laporte et Semet (2000).

Le "Clustering" est une bonne stratégie qui permet de simplifier le travail pour les heuristiques de priorité et ainsi de réduire le temps de résolution. Générer un graphe avec une plus grande granularité et appliquer une heuristique de priorité ne donne pas les mêmes solutions que le "Clustering" qui lui peut agglomérer les tâches d'une façon plus intelligente.

2.4.2 Les Métaheuristiques

Les métaheuristiques sont très efficaces. Elles prennent des solutions, souvent générées par des heuristiques, et elles les améliorent. Leur utilisation sur des problèmes de parallélisation est minimal et se réduit à de simples essais. Ce manque d'application des métaheuristiques est dû à leur complexité et au fait qu'elles soient plus récentes et moins connues des informaticiens. En pratique, on leur préfère des heuristiques simples qu'on peut modifier et adapter plus facilement selon les cas particuliers à résoudre (Ghosh et Gagnon, 1989). Les métaheuristiques les plus connues sont : la recherche avec tabous, les algorithmes génétiques et le recuit simulé.

2.4.2.1 La recherche avec tabous

La recherche avec tabous est l'une des métaheuristiques les plus populaires. Elle produit d'excellents résultats pour la plupart des problèmes combinatoires (voir Glover et Laguna (1997) pour une description détaillée de la recherche avec tabous). C'est un algorithme d'amélioration : elle part d'une solution initiale et essaie de l'améliorer en explorant le voisinage de cette dernière, c'est-à-dire que la solution est modifiée en explorant préalabelement plusieurs changements avant d'en choisir un.

Une bonne recherche avec tabous dépend de la longueur de la liste tabou (ensemble des mouvements interdits) et du voisinage utilisé qui est défini par des mouvements (modifications des composantes de la solution initiale). Les principaux mouvements utilisés sont : (1) le transfert, on déplace une tâche d'un CPU à un autre; (2) et le swap, on échange deux tâches entre deux CPUs. Chiang (1998) a proposé un algorithme basé sur la recherche avec tabous pour le problème d'équilibrage de ligne d'assemblage. Lapierre et Ruiz (2001) proposent une approche flexible de l'utilisation des voisinages : le choix du voisinage est défini dynamiquement selon l'état de la recherche avec tabous.

Le problème d'ordonnancement d'atelier a été également résolu en utilisant des algorithmes de recherche avec tabous par Taillard (1994) et Nowicki et Smutnicki (1996). Thersen (1998) a développé un algorithme de recherche avec tabous pour l'affectation des tâches dans un problème de calcul parallèle. Il s'agit de l'affectation de n tâches indépendantes à m processeurs sans les contraintes ni de communication ni de précédence. La recherche avec tabous reste donc à explorer pour notre problème.

2.4.2.2 Les algorithmes génétiques

Les algorithmes génétiques consistent en l'amélioration d'une population de solutions initiales (la population des parents) pour créer une nouvelle population (la population des enfants). Et pour ce, on se base sur une représentation "génétique" de chaque solution puisqu'on simule le processus d'évolution naturel : croisement génétique, mutation, etc. À la fin, on garde l'élite, soit les meilleures solutions de la population. Cette approche a été utilisée sur plusieurs problèmes combinatoires : Équilibrage des lignes d'assemblage (Rubinovitz et Levitin, 1995); problèmes d'atelier multi-gamme (Lee et al., 1997); problèmes de gestion de projet

avec contraintes de ressources (Mori et Tseng, 1997); problèmes de parallélisation (Vigo et Maniezzo, 1997). Dans ce dernier cas, Vigo et Maniezzo ont appliqué un algorithme hybride, génétique et tabou, à la résolution du problème "Process Parallel Problem". Quoique leur problème inclut des coûts de communication, il ne tient pas compte des contraintes de précédences. Les algorithmes génétiques restent donc à être appliqué à notre problème.

2.4.2.3 Le recuit simulé

Le recuit simulé (Kirkpatrick et al.,1983; Cerny, 1985) combine des idées issues de l'optimisation combinatoire et de phénomènes physiques. Cette métaheuristique peut être perçue comme un algorithme d'amélioration itérative mais, aussi, comme la simulation d'un phénomène physique, à savoir, le processus de chauffage suivi de refroidissement mis en oeuvre pour diminuer les contraintes emmagasinées dans certains matériaux solides sous l'effet de l'irradiation. La principale application de cette approche en ordonnancement est celle faite par Van Laarhoven et al. (1992).

Ces métaheuristiques, ou algorithmes évolutifs, comme la recherche avec tabous ou les algorithmes génétiques produisent de nouvelles solutions à partir de solutions existantes par une certaine opération donnée. Ce processus peut facilement mener à des solutions infaisables, c.-à-d. des solutions qui ne respectent pas les contraintes de précédence, par exemple. Par conséquent, une attention particulière doit être portée sur la définition des voisinages, afin de préserver la faisabilité des solutions, ce qui rend ces métaheuristiques assez difficiles à développer. Il y a beaucoup de travaux de recherche à faire sur le développement des métaheuristiques pour le contexte de calcul parallèle, incluant la parallélisation de l'exécution des heuristiques et métaheuristiques!

2.5 Conclusion

On retient de cette revue de la littérature qu'il y a différentes façons d'aborder la parallélisation soit via le modèle soit via le code de résolution. Nous retenons le contexte mitoyen. Pour le formalisme mathématique on a deux approches soit horaire et par séquence. L'approche par séquence est meilleure mais le modèle résultant reste très difficile à résoudre. C'est pourquoi on trouve toujours dans le calcul parallèle les mêmes heuristiques de bases développées il y a plus de trente ans pour résoudre des problèmes pratiques. Les métaheuristiques, quant à elles, sont très prometteuses mais elles démarrent sur des solutions trouvées par des heuristiques. Ainsi, dans ce travail, nous allons fournir un modèle mathématique pour notre problème et la résolution sera principalement basée sur des heuristiques de priorité et de "Clustering". Nous laissons le développement des métaheuristiques pour de futur travaux de recherche car nous allons devoir travailler très fort sur la génération du DAG. En effet, les travaux sur la génération d'un DAG sur un modèle similaire à celui de Simulink sont inexistants et les ingénieurs d'Opal-rt ne sont pas familiers avec la recherche opérationnelle.

CHAPITRE 3

GÉNÉRATION DU GRAPHE ACYCLIQUE

Comme expliqué dans le chapitre 1, la parallélisation de notre simulation nécessite une phase préalable afin d'identifier les liens entre les tâches à exécuter par les processeurs. Cette phase est critique pour résoudre le problème de parallélisation. On consacre tout ce chapitre à la génération du graphe.

Les simulations concernées par notre projet sont réalisées par un logiciel à interface graphique basé sur un langage de programmation par "diagramme-bloc". La logique très fonctionnelle de ce type de langage permet certes une schématisation bien structurée du code. Toutefois, il faut transformer ces schémas vers un formalisme plus connu de la recherche opérationnelle soit le graphe acyclique orienté - "Directed Acyclique Graph" (DAG).

Partant d'un modèle Simulink tel que montré à la figure 3.1, nous verrons durant ce chapitre comment arriver au DAG le caractérisant, tel que présenté à la figure 3.2. Dans un premier temps, nous présentons la génération des tâches et des liens dans les cas simples puis nous discutons des différents problèmes spécifiques au contexte de nos modèles Simulink. Finalement, nous enchaînons avec les aspects techniques de la génération du DAG.

3.1 Aspects logiques

Dans toute cette section, nous expliquons comment aboutir à un DAG (voir figure 3.2) à partir d'un modèle développé sous Simulink (voir figure 3.1). Nous nous intéressons ici aux aspects conceptuels du modèle Simulink sur lesquels nous allons

nous attarder afin de caractériser les tâches et les liens d'antécédence formant le DAG.

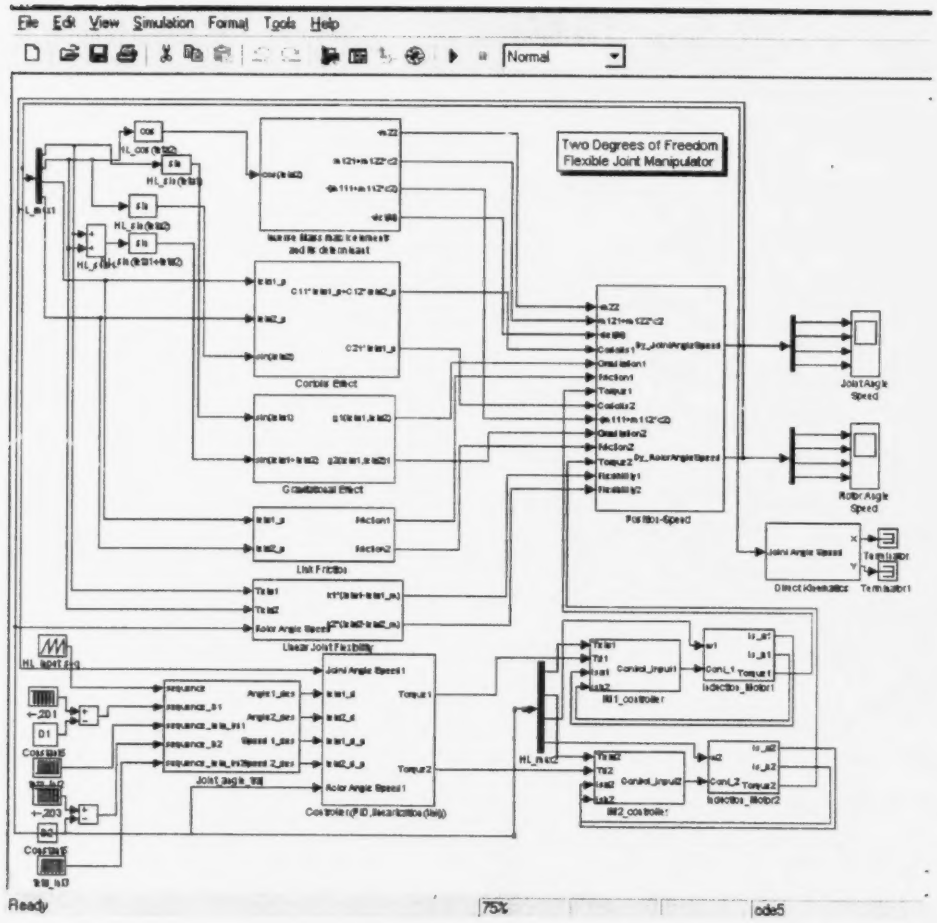

Figure 3.1 : Exemple de modèle Simulink représentant un robot de deux degrés de liberté - un bras avec deux articulations.

Un DAG, comme celui de la figure 3.2, est un ensemble de noeuds (tâches) et d'arcs (liens d'antécédence). Le modèle Simulink de la figure 3.1, quant à lui, est un ensemble de blocs et de flèches contenant des cycles. Alors comment passer de l'un à l'autre? Certes il y a des analogies -schématiques- entre les deux mais il n'en

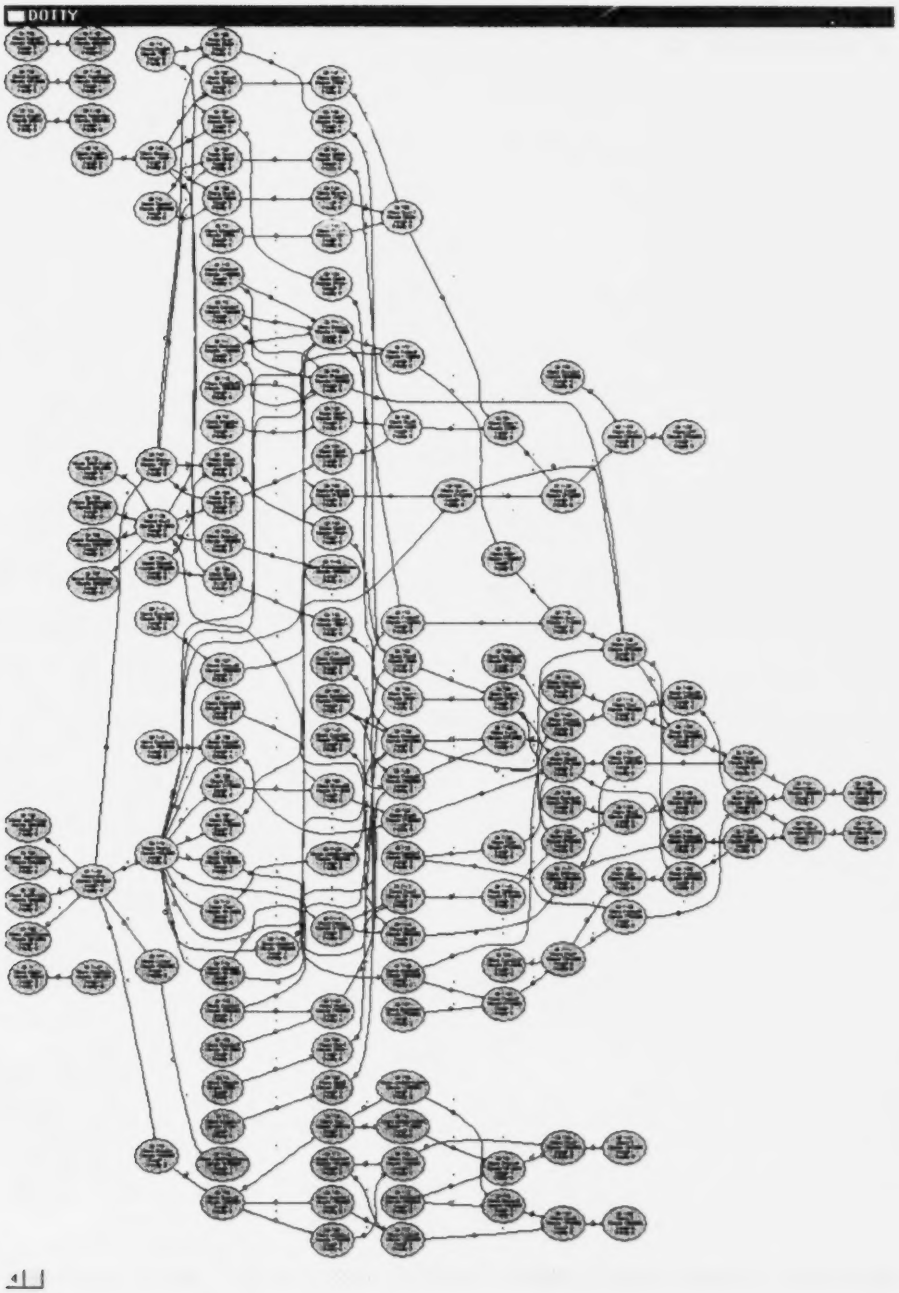

Figure 3.2 : Le DAG correspondant au modèle Simulink de la figure 3.1.

reste pas moins que les deux représentations ne sont pas équivalentes - excepté d'un point de vue fonctionnel. C'est ce qu'on examine dans les sous-sections suivantes.

3.1.1 Des blocs et flèches de Simulink vers les noeuds et les arcs du DAG

Les modèles Simulink sont composés de blocs (triangles, carrés, cercles) et de flèches qui les relient. Les blocs représentent des fonctions qui transforment les variables, d'intrants en extrants, et effectuent ainsi des tâches de calcul. Les flèches représentent des variables physiques ou mathématiques qui sont transmises d'un bloc à un autre pour les besoins de calcul. Elles représentent l'ordre d'exécution du calcul.

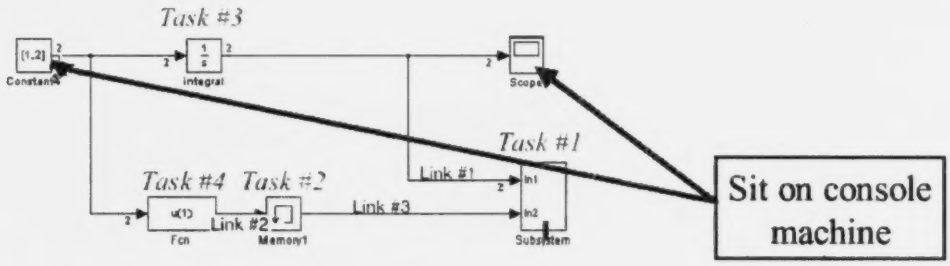

Figure 3.3 : Exemple de correspondance entre Simulink et le DAG. Les blocs d'affichage et d'initialisation "Constant4" et "Scope" ne sont pas considérés comme des tâches.

À priori, on considère que chaque bloc du modèle Simulink est une tâche équivalente à un noeud dans le DAG. Similairement, chaque flèche du modèle Simulink est considérée comme un lien d'antécédence, soit un arc dans le DAG. Ainsi, on a identifié dans la figure 3.3 quatre blocs et trois flèches qui sont directement équivalents à quatre tâches (tasks) et trois liens d'antécédence (links) d'un DAG. Toutefois cer-

tains blocs de cette figure ne représentent aucune transformation ni calcul. Ils sont là uniquement pour embellir le modèle où représentent des tâches non propres à la simulation. Ainsi, à la figure 3.3, le bloc "Constant4" contenant des variables d'initialisation, et le bloc d'affichage (ou de représentation des résultats) "Scope" ne représentent aucun calcul. Ces blocs seront omis dans le DAG sans perte de logique. Ils sont mis à part dans la console de contrôle de la simulation gérée par Rt-lab.

3.1.1.1 Blocs et tâches - noeuds

Après un travail préliminaire de tri entre ce qui est vraiment une tâche et les blocs d'habillage, chaque bloc de Simulink donne naissance à une tâche (un noeud) pour le DAG définie par : le **nom**, le même que celui sur le modèle; la **taille**, durée de calcul associée au bloc; et le **numéro ID** (handle) qui est l'identificateur unique de chaque élément dans Simulink - ce dernier nous permettra une relation directe entre le graphe et le modèle.

3.1.1.2 Flèches et liens d'antécédence - arcs

Une fois les tâches définies, toutes les flèches entre les blocs associés à des tâches sur le modèle du DAG donnent naissance à des arcs définis par : le **numéro ID** (handle), un identificateur unique et la **taille**, somme des variables transitantes entre les deux tâches, converties en "bit" selon leur type (un "double" = 64 bit).

Ainsi, chaque bloc et chaque ligne du modèle Simulink correspondent à une entité du DAG. Cependant la finesse ou le détail défini par le premier niveau du modèle n'est forcément pas le meilleur pour obtenir un bon graphe. En effet la structure hiérarchisée du modèle nous oblige à considérer la granularité du graphe.

3.1.2 Granularité du graphe

L'univers de Simulink est complexe. Chacun des blocs peut contenir un sous-système contenant lui-même d'autres sous-systèmes. En effet, dans l'environnement de Simulink, l'utilisateur construit son modèle de simulation en reliant des blocs obtenus soit à partir des librairies Simulink (NBS) soit à partir des blocs personnalisés. Ainsi on trouve quatre types de blocs dans un modèle de Simulink :

- 1. Bloc indigène de Simulink (NBS): c'est le bloc de base représentant une opération élémentaire, Ssa taille est de l'ordre de la nanoseconde;
- 2. S-fonction : bloc permettant d'intégrer des librairies externes sous forme de "dll" ou code "C";
- 3. Bloc comprenant le code de MATLAB (ce dernier type de bloc n'est pas supporté par la génération de code avec RTW);
- 4. Sous-système : sous-ensemble composé d'autres blocs selon des critères propres au concepteur. C'est comme un petit modèle Simulink. Sa taille est variable et de l'ordre de un à cent microsecondes. Ces sous-systèmes personnalisés peuvent être masqués ou non (selon que le concepteur accepte qu'on les explore ou non).

Les trois premiers types de blocs ne peuvent pas être éclatés mais le quatrième type, le sous-système, est un "super bloc" qui peut être considéré comme une seule tâche dans notre DAG ou un ensemble de blocs et de flèches (tâches et liens d'antécédence).

Puisque le modèle Simulink est hiérarchique, les sous-systèmes risquent de générer des DAG déséquilibrés s'ils ne sont pas considérés à un niveau de détail plus élevé car la taille des sous-systèmes est souvent de l'ordre de mille fois celle des NBS.

Donc, si on éclate les sous-systèmes, on peut obtenir un graphe plus équilibré mais on risque de générer des graphes de très grande taille. Pour l'exemple de la figure 3.1, qui est un petit modèle, le graphe le plus fin est composé de plus de 900 tâches. Pour une affectation manuelle des tâches aux différents CPU (pour le calcul parallèle), un ingénieur ne peut évidemment pas s'occuper des 900 tâches. Pour nos algorithmes, un graphe de 900 tâches est problématique pour des heuristiques simples qui sont moins performantes s'il y a trop de détail, comme nous le verrons dans un chapitre ultérieur.

Afin de trouver un DAG avec le niveau de détail adéquat, nous avons besoin du concept de granularité. Nous l'utilisons de la façon suivante : (a) nous définissons une taille critique à partir de laquelle nous décidons d'explorer ou non un sous-système; (b) nous parcourons alors le modèle Simulink en éclatant seulement les sous-systèmes excédents cette limite, en rétablissant les liens des blocs au niveau supérieur comme il se doit; (c) finalement, afin d'obtenir un graphe plus équilibré et plus compact, nous effectuons un nouveau balayage du graphe pour regrouper les tâches consécutives sur la même branche en une seule. Avec le concept de granularité, nous pouvons générer plusieurs graphes pour un même modèle, en faisant varier le paramètre de la taille critique. Ceci est très utile pour trouver le bon niveau de détail pour nos heuristiques d'affectation de tâches. C'est également essentiel pour l'ingénieur voulant vérifier la qualité des solutions proposées, qui peut faire varier les solutions de parallélisation de notre heuristique en variant la granularité. À noter que notre implantation de la granularité est différente de celle expliquée dans les livres de calcul parallèle où la granularité est définie par la taille de la plus petite tâche dans un graphe, et non par la plus grande tâche acceptée dans le graphe.

Cette technique appliquée au modèle de la figure 3.3 donne, après éclatement du bloc "Subsystem", le modèle et le graphe illustrés à la figure 3.4, qui contient cinq

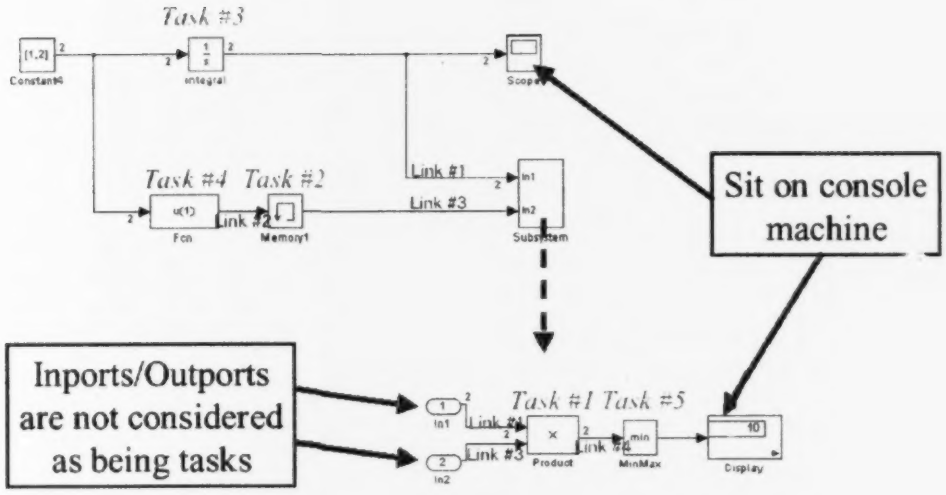

Figure 3.4 : Exemple de correspondance entre Simulink et le DAG après éclatement des sous-systèmes.

tâches plutôt que quatre.

3.1.3 Modèle cyclique

Les modèles Simulink peuvent contenir des cycles : soit parce que certains systèmes sont eux-mêmes cycliques, comme ceux faisant intervenir la rétroaction (voir exemple de la figure 3.7), soit parce que le concepteur du modèle les a fait intervenir pour faciliter la compréhension du modèle. Par opposition au modèle, le DAG doit être acyclique car nos algorithmes d'ordonnancement ne peuvent pas traiter les graphes cycliques.

Le caractère acyclique du DAG est indispensable pour la suite de notre projet d'une part, et d'autre part, après investigation du modèle cyclique dans Simulink, nous avons trouvé qu'il est informatiquement impossible de modéliser le calcul d'un

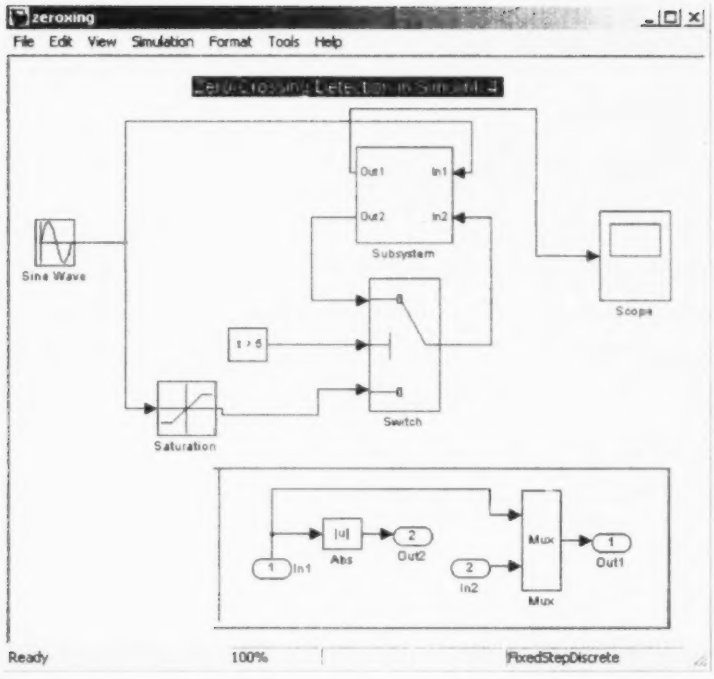

Figure 3.5 : Exemple d'un faux modèle cyclique (le cycle est formé entre les blocs "Subsystem" et "Switch"). Le rectangle en bas montre le contenu du sous-système "Subsystem".

cycle sans introduire des artifices. Ainsi, les modèles cycliques doivent toujours être traités par un modèle ou une méthode afin de rendre le calcul acyclique. Pour mieux comprendre ce mécanisme, on regarde l'origine de ce caractère cyclique dans Simulink pour trouver comment en extraire un graphe acyclique.

3.1.3.1 Origine et nature du caractère cyclique

Le caractère cyclique d'un modèle Simulink peut être de deux natures : (1) le concepteur du modèle, par souci de clarté et de lisibilité du schéma, introduit intentionnellement des boucles cycliques qui alors induisent un faux modèle cyclique; (2) le phénomène modélisé est naturellement cyclique et nous avons alors un modèle vraiment cyclique.

Pour le premier cas, qui est le plus simple conceptuellement, son origine peut être simplement une manipulation de la part du modélisateur qui, en regroupant des blocs en un sous-système, crée une boucle au niveau supérieur. L'exemple de la figure 3.5 est un modèle Simulink où nous avons expressément regroupé les blocs "Abs" et "Mux" - que nous voyons au bas de la figure – en un sous-système "Subsystem". Dans ce modèle nous avons une boucle formée par les blocs "Subsystem", "Switch" et les liens qui les lient. Mais si nous éclatons le sous-système puis nous amenons le tout au premier niveau, comme on le présente à la figure 3.6, nous constatons que le modèle est parfaitement acyclique.

Pour le deuxième cas, le phénomène modélisé derrière est naturellement cyclique. Les systèmes à rétroaction, tel qu'illustrés à la figure 3.7 est de nature cyclique. Le modèle de la figure 3.8, un pendule simple, est aussi cyclique car la position angulaire "tetha" dépend de la vitesse angulaire "tetha dot" qui dépend à son tour de la force exercée sur le pendule (son propre poids et le moment extérieur). Mais

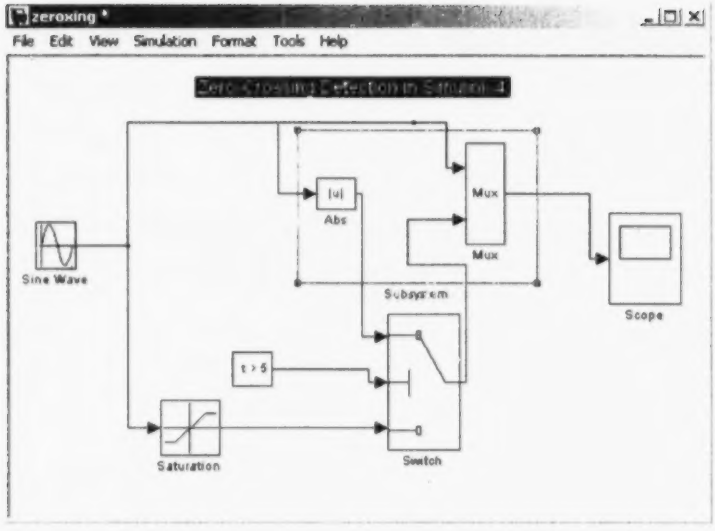

Figure 3.6 : Le modèle de la figure 3.5 une fois éclaté et ramené à un seul niveau. On constate l'absence de la boucle cyclique

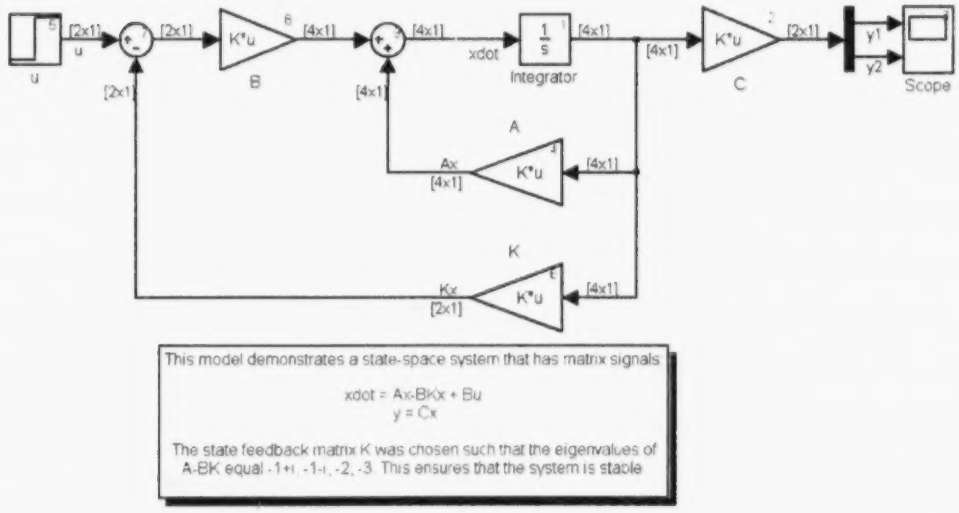

Figure 3.7 : Exemple d'un système rétroactif contenant un cycle.

cette dernière dépend aussi de "tetha" et de "tetha dot", ce qui crée un cycle.

L'origine des cycles est donc soit due à la construction du modèle soit au phénomène modélisé qui est proprement cyclique. Dans la partie qui suit nous allons voir comment enlever ces cycles.

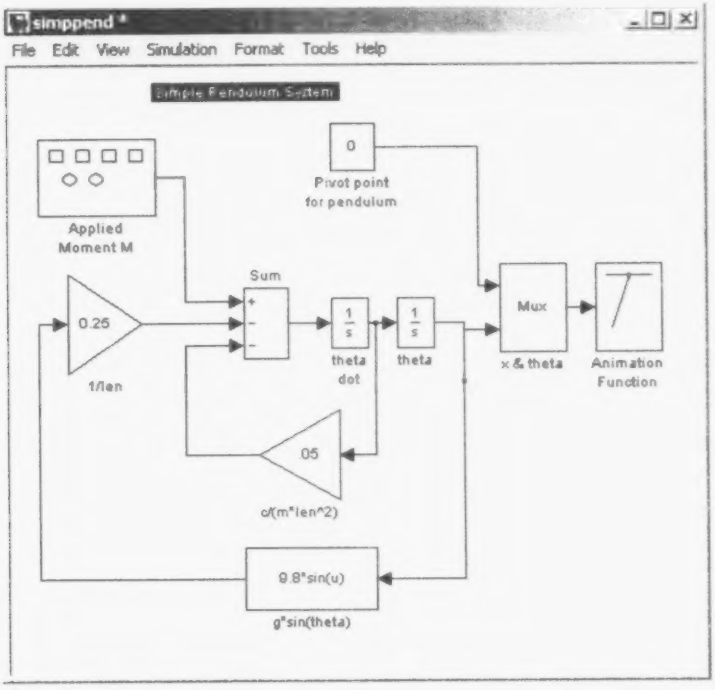

Figure 3.8 : Exemple d'un vrai modèle cyclique : un pendule simple.

3.1.3.2 Modélisation proposée pour rendre le graphe acyclique

Puisque le DAG est acyclique, il est très important de trouver une logique pour défaire les boucles dans Simulink. En ce qui concerne le cas où les cycles sont introduits par le concepteur, la solution est conceptuellement simple. Il suffit d'annuler les manipulations du modèle à l'origine des cycles : il faut décortiquer tout le modèle

en enlevant tous les sous-systèmes où les blocs risquent d'induire des cycles. Pour cela, il suffit d'éclater le modèle jusqu'à la granularité la plus fine. Malheureusement ceci n'est pas optimal car, si nous sommes en présence d'un modèle complexe à plusieurs niveaux, ceci demande beaucoup de temps de manipulation du graphe. Donc, comme compromis, nous optons pour la méthode qui suit : (a) comme vu dans la section sur la granularité (la section 3.1.2), seuls les sous-systèmes dont la taille excède le poids critique seront éclatés; (b) puis, si nous détectons un cycle [1] lors de la construction du DAG, nous retournons au sous-système en cause dans Simulink, et nous essayons de l'éclater.

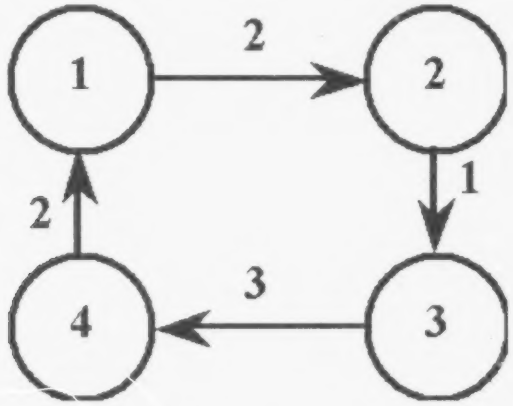

Figure 3.9 : Un exemple de cycle

Dans le deuxième cas où le cycle est naturel, une étude plus approfondie des blocs de Simulink s'impose. Tout d'abord un cycle ne peut nullement être simulé sans introduire une imprécision au calcul du modèle. Par exemple, pour le cycle de la figure 3.9 représentant un cas de rétroaction, le logiciel de simulation impose de commencer le calcul par l'initialisation d'une tâche donnée avec une valeur

[1] On peut détecter facilement les cycles d'un DAG en triangularisant la matrice d'adjacence. Dans notre cas, vu la nature du problème, la triangularisation de la matrice d'adjacence est très facile car les vecteurs de base restent les mêmes : on ne fait que des permutations sur ces derniers. Un algorithme simple proche de la logique du "QuickSort" est suffisant pour trianguläriser.

arbitraire : le cycle est donc "linéarisé". Ainsi, notre cycle de la figure 3.9 est traité comme sur la figure 3.10, c'est-à-dire comme si c'était une chaîne débutant par la tâche (1) et finissant par la tâche (4). Quant au lien entre (1) et (4), la valeur est transmise seulement à la fin d'une itération pour débuter l'itération de l'étape suivante. Cette méthode est couramment référée à l'introduction d'un "retard" : à l'itération "n", la tâche (1) utilise l'extrant de la tâche (4) obtenu à la fin de l'itération "n-1".

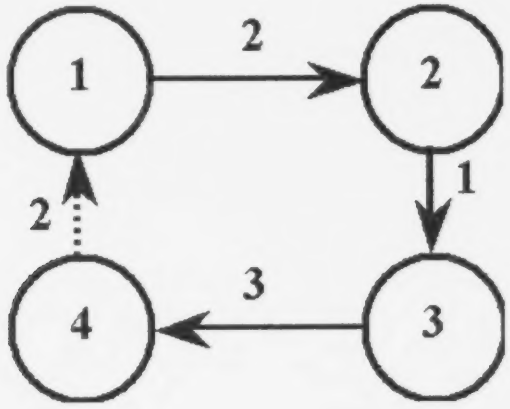

Figure 3.10 : Un exemple de traitement d'un cycle. Le lien 4 – 1 est retardé.

D'après nos analyses, Simulink utilise le principe de linéarisation avec les retards. Avant de proposer notre solution à ces cycles, on présente des caractéristiques de Simulink pertinentes à sa compréhension.

3.1.3.3 La représentation des phénomènes cycliques dans Simulink

Simulink, comme les autres logiciels de simulation, introduit des retards dans les cycles pour les calculer. Dans Simulink, on peut classifier les variables transitant

sur les lignes dans un modèle en deux types [2] :

Les "Variables" : elles représentent les sorties des blocs qui dépendent directement de l'entrée et qui sont transmises durant la même itération de la simulation. Elles transitent d'un bloc à l'autre et soulignent ainsi un vrai lien de précédence. Ainsi, ce sont des variables qui sont modifiées ou mises à jour puis transmises au bloc suivant durant le même pas de simulation.

Les "États" [3] : sorties des blocs qui ne dépendent pas directement de l'entrée et qui sont transmises entre les itérations. Ils ne représentent Donc aucun vrai lien de précédence. Ainsi, ce sont des variables qui sont modifiées ou mises à jour durant un pas de simulation puis transmises aux blocs suivant à l'itération suivante.

Puisqu'il y a deux types de variables, par conséquent, il y a deux types de blocs comme indiqué sur la figure 3.11, ou vice-versa. Tous les blocs NBS sont classifiés par Simulink en deux catégories et on a accès à cette information directement. Les deux catégories sont :

Les blocs "FeedThrough" Blocs qu'on peut appeler "à transmission directe" car l'extrant est fonction directe de l'intrant de la même itération. C.-à-d. ce sont des blocs qui envoient des variables "Variables" (voir ci-dessus section : 3.1.3.3).

Les blocs "Non-FeedThrough" Ceux-ci peuvent être appelés blocs "à transmission différée" car l'extrant n'est fonction que de l'intrant de l'itération

[2] Cette classification et la terminologie présentée proviennentt des ingénieurs de la compagnie Opal-rt.

[3] États : à ne pas confondre avec l'état d'un système. Quoique ce dernier reflète l'état, mesure caractéristique du bloc durant un pas. Pour notre contexte, il s'agit de variables transmises durant un pas de simulation i pour donner la mesure du pas d'avant i-1.

d'avant. C.-à-d. ce sont des blocs qui envoient des variables "États" (voir ci-dessus section : 3.1.3.3).

Dans l'exemple de la figure 3.11, on peut constater les différents types de blocs et de liens suivants :

- Les blocs "Gain" sont des blocs "FeedThrough";
- Le bloc "Integrateur" est un bloc "Non-FeedThrough";
- Des "États" transitent sur le lien "Link 2";
- Des "Variables" transitent Sur les liens "Link 1" et"Link 3".

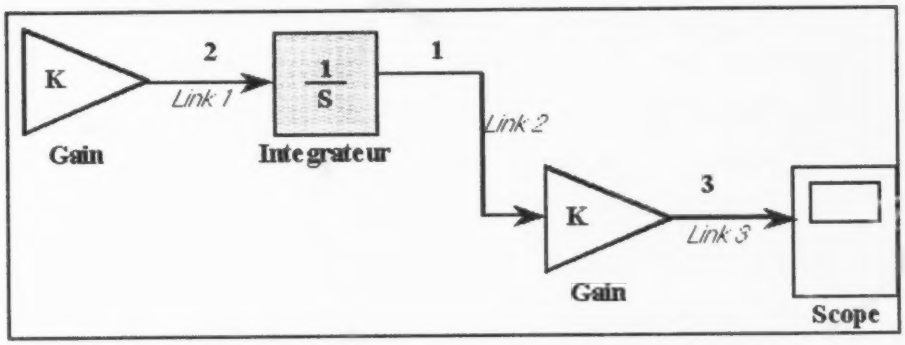

Figure 3.11 : Exemple de blocs "FeedThrough" (Gain) et "Non-FeedThrough" (Integrateur).

3.1.3.4 Le fonctionnement des blocs "Non-Feedthrough"

Dans l'environnement Rt-lab, un bloc "Non Feedthrough" du modèle Simulink est généralement exécuté de la manière suivante :

- *Initialisation des variables* : Puisqu'on introduit généralement un retard dans un bloc Non-Feedthrough, la première itération sert à initialiser les entrées pour démarrer le calcul. Ceci se fait une seule fois au début de la simulation car, dans les autres itérations, les intrants du blocs proviennent de l'itération d'avant;

- *Calcul des extrants (OutPuts)* : Cette partie-là est celle qui fait les calculs nécessaires aux blocs successeurs. C'est seulement cette partie-là qui doit précéder les blocs suivants, car ces calculs utilisent les états du bloc de l'itération d'avant. Elle n'a donc pas de bloc prédécesseur dans l'itération présente;

- *Mise à jour des états (States Update)* : Cette partie utilise les intrants de la même itération pour mettre à jour l'état du bloc. Elle dépend donc des prédécesseurs du bloc mais n'a pas de successeurs dans la même itération.

Donc, un bloc "Non-Feedthrough" se comporte comme deux blocs de calcul indépendants dans la même itération. Simulink considère ces deux tâches dans le même bloc mais pour notre DAG, il faudra les scinder en deux.

3.1.3.5 Représentation des blocs "Non-Feedthrough"

En se basant sur la description ci-dessus, un bloc "Non-Feedthrough" doit être représenté par deux tâches distinctes : (1) Une tâche pour les *"OutPuts"* qui dépend des tâches prédécesseurs du bloc; (2) et une tâche pour les *"States Update"* qui fournit les sorties pour les tâches successeurs du bloc.

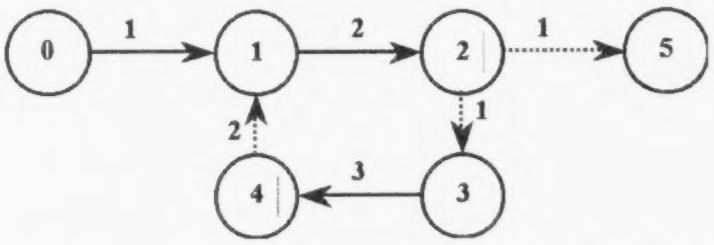

Figure 3.12 : Exemple de modèle cyclique avec des tâches "Non-Feedthrough" (les tâches grisées).

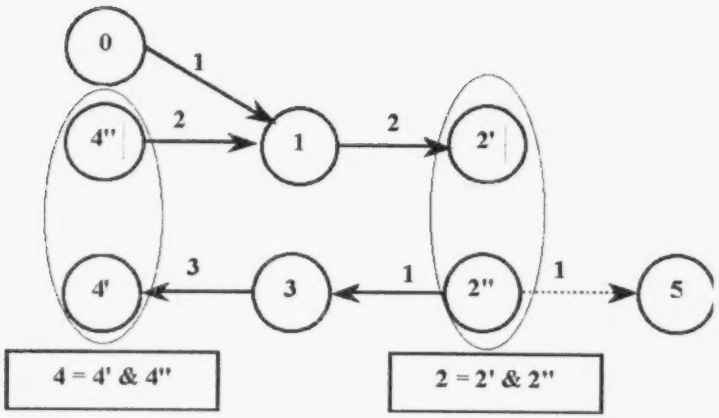

Figure 3.13 : Le modèle équivalent à l'exemple de la figure 3.12 du point de vue de l'exécution sous Rt-lab.

D'après le traitement de Rt-lab, le modèle de la figure 3.12 est équivalent au modèle représenté sur la figure 3.13 où le bloc "Non-Feedthrough" est remplacé par deux blocs. À noter que chaque bloc "Non-Feedthrough" (les tâche 2 et 4) est représenté par deux tâches distinctes avec le même numéro et un attribut différent (2', 2", 4', 4").

Les deux tâches issues d'un bloc "Non-Feedthrough" sont indépendantes au niveau du DAG mais elles doivent être exécutées sur le même CPU car représente la même entité. Ainsi, nous appelons ces deux tâches des "**Co-tâches**". De même, le modèle Simulink de la figure 3.7 est équivalent au modèle sans boucle montré à la figure 3.14 car le bloc "Integrator" est scindé en deux. Sur cette dernière figure, nous avons indiqué l'ordre d'exécution réel des tâches obtenu sous Rt-lab. Ceci démontre que notre modélisation de la logique Simulink est correcte. C'est cette représentation que nous adoptons pour la suite de nos travaux pour transformer les cycles.

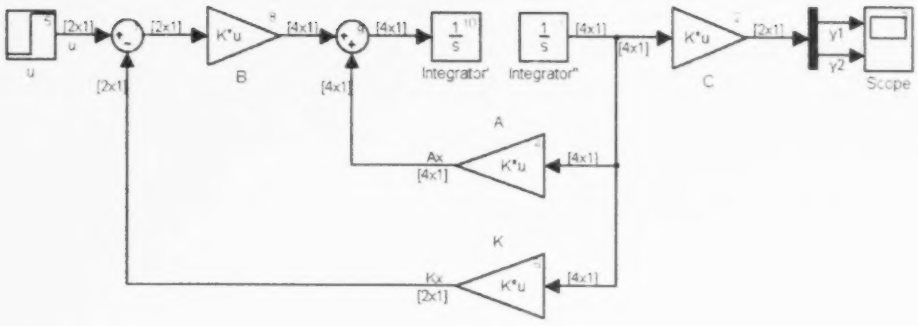

Figure 3.14 : Le modèle équivalent à l'exemple de la figure 3.7 du point de vue de l'exécution sous Rt-lab. avec l'ordre d'exécution des tâches (il est indiqué par les chiffres en haut et à droite des blocs.)

3.1.3.6 Conclusion concernant le graphe acyclique

Les cycles dans notre modèle Simulink sont de deux natures, soit ceux dus aux regroupements de blocs faits par le concepteur soit ceux dus à la nature même du phénomène modélisé. Pour ceux dus au concepteur la solution est de jouer sur la granularité du graphe, afin de détecter la majorité des cycles, les autres étant éliminés en éclatant tout le sous-système. Par contre, pour ceux dus à la nature du phénomène, on distingue les blocs "Feedthrough" des blocs "Non-Feedthrough" (l'information est directement accessible de Matlab). Les blocs "Feedthrough" sont gardés tels quels dans le DAG. Par contre, les blocs "Non-Feedthrough" sont représentés par deux tâches au lieu d'une, permettant de briser les cycles. On les appelle "Co-tâches" car on exige qu'elles soient exécutées sur le même CPU.

3.2 Aspects techniques de l'automatisation

Maintenant que nous avons établi les règles pour la génération du DAG, nous présentons le côté technique de la génération automatique du DAG.

Comme énoncé dans le chapitre 1, Matlab offre des fonctionnalités qui permettent le contrôle des modèles Simulink. Nous disposons de deux façons de manipuler un modèle. La première est d'utiliser le fichier ".mdl" (voir chapitre 1). En parcourant ce fichier "ASCII", nous pouvons détecter la composition objet du modèle. Et si nous modifions le contenu du fichier mais en respectant son architecture, nous pouvons modifier le modèle Simulink. La deuxième façon est d'utiliser les fonctions du descripteur de Matlab. À l'aide de ce "Script m" [4] nous pouvons manipuler le modèle de l'extérieur de Simulink. Matlab offre des accès (handles) qui permettent d'accéder et de manipuler directement les composantes du modèle. Nous avons choisie cette approche du "Script m" pour implémenter notre algorithme de génération du DAG.

Nous terminons ce chapitre par une description de notre algorithme générant le DAG suivie d'un exemple de son fonctionnement.

3.2.1 Description de l'algorithme de génération du DAG

L'algorithme de génération du DAG utilise un modèle Simulink et retourne les informations relatives au DAG dans une matrice d'adjacence ainsi qu'une table (structure) décrivant les attributs des tâches :

La matrice d'adjacence : C'est une matrice carrée de taille $n \times n$. Les indices i

[4] Le "Script m" est le langage de programmation utilisé dans l'environnement Matlab. C'est un script car il est directement interprété par la machine.

et j sont les numéros des tâches. La matrice contient comme valeurs : zéro s'il n'y a pas de lien entre la tâche i et la tâche j; le poids de communication s'il y a un lien. De plus, si le lien est "Non-Feedthrough", le poids de la communication est précédé d'un signe négatif. Nous avons choisi cette représentation matricielle car elle est plus facile à manipuler dans le contexte matlab. À noter également que nous avons opté pour l'ajout d'un négatif pour identifier les tâches "Non-Feedthrough" plutôt que de dupliquer les tâches dans la matrice. C'est l'algorithme de résolution qui s'occupera de la duplication des co-tâches dans le graphe.

La table des attributs des tâches : Cette table fournit différents attributs qui servent à décrire les tâches : taille (en durée de calcul), nom, type, "handle" (identificateur unique sous Simulink) et autres informations permettant la restitution du modèle.

La figure 3.15 montre la division du code de génération du DAG en "Script m". L'algorithme contient trois phases : (I) caractérisation des tâches et des liens; (II) génération du DAG, soit la matrice d'adjacence; et (III) création de la table des attributs des tâches. Aussi, comme indiqué sur la figure 3.15, on a une dernière phase qui sert à visualiser le DAG en utilisant "GraphViz", un utilitaire de visualisation des graphes. Les trois phases de notre algorithme sont brièvement décrites dans ce qui suit.

3.2.1.1 Caractérisation des tâches et des liens d'antécédence

Le but de cette première phase est de parcourir tout le modèle Simulink et de caractériser ses composantes avant de créer le DAG. Ceci est important car, d'une part, nous ne connaissons pas la profondeur du modèle - qui est constitué de plusieurs

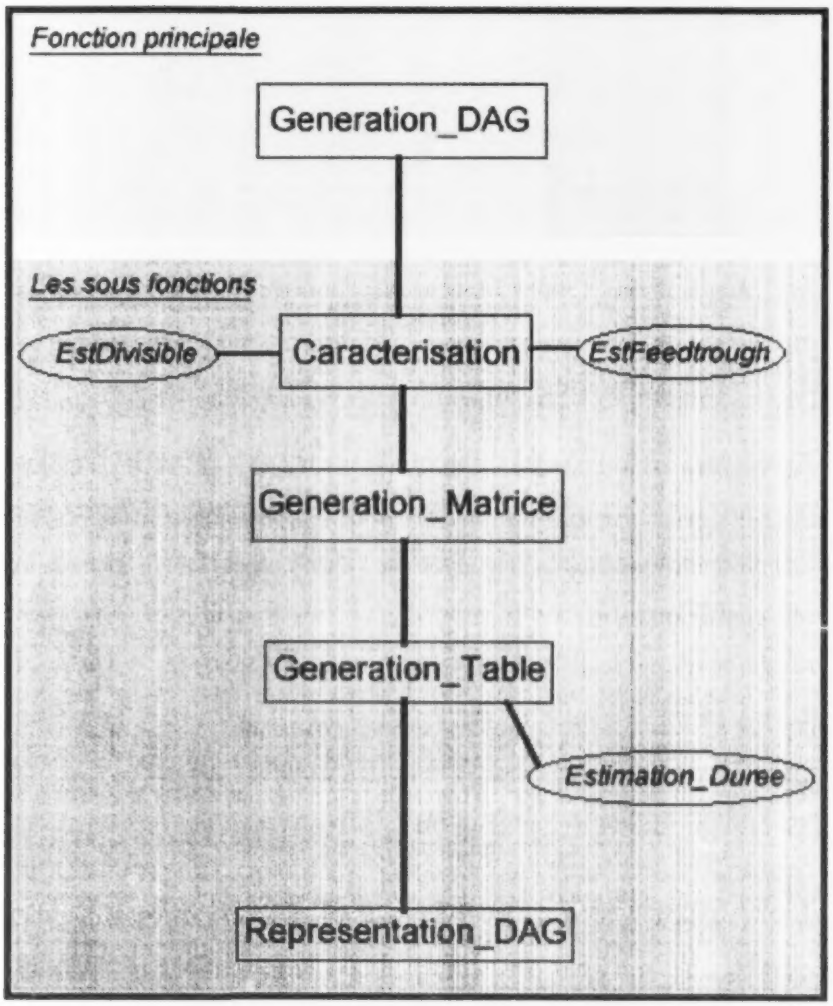

Figure 3.15 : Algorithme de génération du graphe "DAG"

niveaux - et d'autre part, certains sous-systèmes doivent rester inséparables (ceci permet au concepteur de décider de garder un ensemble de tâches sur un même CPU). Cette fonction ouvre le modèle sous l'environnement Matlab et, grâce aux "handlers" (gestionnaires) de ce dernier, elle fait une première classification des blocs entre les NBS et les sous-systèmes. Tous les NBS sont alors déclarés comme tâches exceptés ceux d'habillage d'affichage ou d'initialisation de constantes. Les sous-systèmes, quant à eux, sont traités comme suit : s'ils sont masqués par le concepteur (tous les sous-systèmes du type "enabled", "triggered" ou "masked") ou que leur taille (en termes de durée de calcul) est inférieure à la taille critique de la granularité du DAG, ils sont conservés et déclarés comme des tâches. Sinon, ils sont traités par cette même fonction une fois le bloc éclaté. Par la suite toutes les flèches (liens) dans le modèle Simulink sont analysés afin de savoir s'ils sont des liens "Feedthrough" ou "non-Feedthrough". Le pseudocode de la fonction "Caracterisation" présenté à la figure 3.16. À noter que la matrice d'adjacence et la table d'attribut des tâches ne sont pas créées par la fonction "Caracterisation", mais plutôt par les deux fonctions suivantes.

Dans le pseudocode de la figure 3.16, la fonction "Caracterisation" ouvre le modèle <modele_Simulink> - qui est le modèle à représenter sous forme de DAG - et met son contenu dans la variable objet <Mon_Systeme>. Ensuite et grâce aux "handlers" de Matlab on parcourt tous les blocs <Element> de <Mon_Systeme> afin de savoir s'ils sont des NBS ou des sous-systèmes. Alors, si <Element> est un NBS, il est directement défini comme une tâche (qui va figurer dans le DAG) en assignant à la proriété <.EstTache> de ce dernier la valeur <Oui>. Par contre si <Element> est un sous-système alors on appelle la fonction "EstDivisible" qui va vérifier si le sous-système sera éclaté ou non : un sous-système est éclaté si sa taille est plus grande que le seuil de granularité défini ou si il est défini comme "enabled", "triggered" ou "masked". Ainsi, si <Element> est divisible alors on appelle la

```
fonction Caracterisation(modele_Simulink)
    ouvrir modele_Simulink
        {sous matlab la meme fonction sert pour ouvrir
        un modele ou un sous-systeme }
    Mon_Systeme <- modele_Simulink
        {seulement le nivau superieur est visible}
    pour_chaque Element dans Mon-Systeme
        {Element est ou un bloc ou un sous-systeme}
        si Element est un NBS
            Element.EstTache <- oui
            pour_chaque Lien dans Tache.outports
                Lien.EstFeedthrough <- EstFeedthrough(Element)
            fin pour_chaque
        si Element est un SousSysteme
            si (EstDivisible(Element))
                Mon_Systeme <- Element
                caracterisation(Mon_Systeme)
            sinon
                Element.EstTache <- oui
                pour_chaque Lien dans Tache.outports
                    Lien.EstFeedthrough <- EstFeedthrough(Element)
                fin pour_chaque
            fin si
        fin si
    fin pour_chaque
fin fonction
```

Figure 3.16 : Pseudocode pour la caractérisation des tâches et des liens d'antécédence.

même fonction "Caracterisation" avec comme argument l'objet <Element> (en effet, sous Matlab un sous-systeme d'un modèle Simulink est considéré lui même comme un modèle et on peut lui appliquer les mêmes traitements). Par contre, si <Element> est non divisible, il est déclaré comme une tâche, comme dans le cas des NBS. En même temps, chaque fois qu'on rencontre un <Element> qui est une tâche, on parcourt tous les liens (flèches) du modèle Simulink reliés à cette tâche grâce à la proriété ".outports" directement accessible par les "Script m". Puis avec la fonction "EstFeedthrough" on vérifie si la tâche est "Feedthrough" ou non, ce qui permet de caractériser les liens comme "Feedthrough" ou "non-Feedthrough". Ainsi, à la fin de cette fonction on a toutes les tâches qui seront créées dans le DAG. Il ne reste qu'à définir la matrice d'adjacence et la table des tâches.

3.2.1.2 Génération du DAG - de la matrice d'adjacence

Une fois que le modèle Simulink et ses composantes sont caractérisés, on passe à la création de la matrice d'adjacence avec les tâches définies par la procédure précédente "Caractérisation". Ces tâches sont analysées plus en détails au niveau de ses tâches antécédentes afin de remplir la matrice d'adjacence. Plus particulièrement, le temps de communication doit être estimé si les deux tâches sont affectées sur deux CPU différents et un signe négatif doit être ajouté à sa valeur si le lien est "non-Feedthrough". Le pseudocode est présenté à la figure 3.17.

La figure 3.17 illustre le pseudocode de la fonction "Generation_Matrice" qui récupère l'objet <Modele_Simulink_Caracterise>, le modèle Simulink déjà caractérisé par la fonction "Caracterisation". Puis pour chaque tâche de cet objet on détecte les <Liens> émanants de cette dernière par la propriété ".outports". Ensuite, on determine pour chaque lien la <Tache_Destination> (la tâche adjacente), sa taille de <Communication> grâce à la propriété ".size". Enfin, on remplie la matrice

```
fonction Generation_Matrice(Modele_Simulink_Caracterise)
    pour_chaque Tache dans Modele_Simulink_Caracterise
        i <- Tache.index
        Liens <- Tache.outports
            {''outports'' est attributs dans chaque model
            qui donne directement tous les lignes sortantes
            d'un bloc ou sous-systeme}
        pour_chaque Lien dans Liens
            Tache_Destination <- Lien.destination
                {connaissant le lien, la propriete
                ''destination'' sous matlab donne acces
                directement a la tache vers laquelle part
                le lien.}
            j <- Tache_Destination.index
            si (Lien.EstFeedthrough)
                Communication <- Lien.taille
                    {''size'' une propriete qui donne
                    acces a la taille des variables
                    transitant sur un lien.}
            sinon
                Communication <- (-1) * Lien.size
            fin si
            Matrice_Adjacence(i, j) <- Communication
        fin pour_chaque
    fin pour_chaque
fin fonction
```

Figure 3.17 : Pseudocode pour la génération de la matrice d'adjacence.

d'ajacence <Matrice_Adjacence> ou i est l'indice de la variable <Tache> d'origine, j est l'indice de la variable <Tache_Destination> (la tâche adjacente) et la valeur du temps de communication inter CPU est donnée par <Communication> multipliée par (-1) si le lien est "non-Feedthrough" (propriété qui a été identifiée par la fonction "Caracterisation").

3.2.1.3 Création de la table des attributs des tâches

Le but de la dernière phase est de réccupérer quelques informations additionnelles sur la tâche qui seront nécessaires pour l'exécution de l'algorithme d'ordonnancement et pour pouvoir reconstruire le modèle Simulink avec les informations de parallélisation une fois l'algorithme complété. La "Generation_Table" est assez simple. Elle parcourt le modèle caractérisé et enregistre les informations nécessaires. Cependant la sous-fonction qui estime la durée des tâches est très complexe car elle consiste en la quantification de la durée de chaque instruction constituant la tâche. Le pseudocode de la figure 3.17 résume la fonction. de la génération de la table des tâches. Elle prend en entrée l'objet <Modele_Simulink_Caracterise> et puis pour chacune des tâches de cet objet, enregistre dans la <Table_Taches> les informations nom, handle, taille, etc... La taille d'une tâche est determinée par la fonction "Estimation_Duree" qui parcourt toutes les instructions de la tâche afin de quantifier sa durée. Une alternative serait d'exécuter la tâche pour avoir sa durée mais cette procédure s'avère trop longue et trop complexe à faire dans le contexte de Simulink.

```
fonction Generation_Table(Modele_Simulink_Caracterise)
    pour_chaque Tache dans Modele_Simulink_Caracterise
        i <- Tache.index
        Table_Taches(i).nom = Tache.nom
        Table_Taches(i).handle = Tache.handle
        Table_Taches(i).taille = Estimation_Duree(Tache)
        ... etc
    fin pour_chaque
fin fonction
```

Figure 3.18 : Pseudocode pour la génération de la table des tâches.

3.2.2 Exemple d'application de l'algorithme de Génération du DAG

L'exemple ci-dessous illustre l'exécution de notre algorithme en générant le DAG du modèle Simulink "test-DAG" de la figure 3.19. Ce modèle inclut presque tous les éléments importants : un sous-système, soit le bloc "SubSystem" dont le détail est illustré dans la même figure 3.19, des blocs "non-Feedthrough", soient les blocs "Memory" et "Integral", et des blocs "Feedthrough". Les valeurs indiquées sur les flèches de ce modèle sont les temps de communication.

Premièrement l'algorithme génère la structure de la matrice d'adjacence montrée à la figure 3.20 mais on ne la rempli pas. Pour ce, nous avons fixé un degré de granularité de sorte que le sous-système "SubSystem" soit éclaté par l'algorithme. Dans la figure 3.20 on voit que l'entête des lignes et des colonnes de la matrice sont exactement les noms des tâches qu'on voit sur le modèle de départ de la figure 3.19. Bien sûr le bloc "SubSystem" n'apparaît pas sur la matrice car on a choisi d'éclater ce sous-système. Par contre, les blocs composant le sous-système y figurent bien. Ensuite, nous définissons toutes les tâches et les liens entre elles avec leur taille (ici l'unité pour la taille des liens de communication est le nombre de variables transmises qui sont en général, de type "double"). Les données

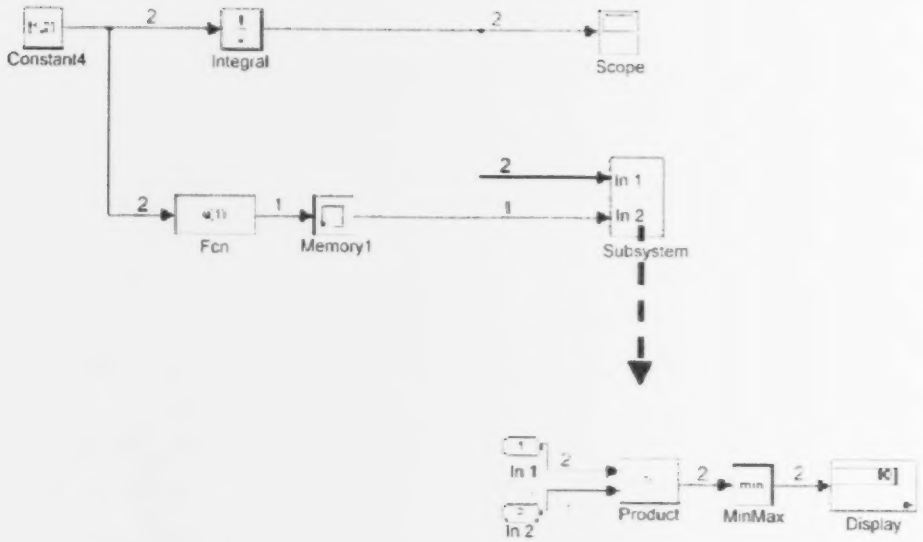

Figure 3.19 : Le modèle Simulink "test-DAG"

sur les tâches sont stockées dans des structures internes à Matlab pour accélérer l'exécution de l'algorithme mais elles sont récupérées à la fin pour générer la table de définitions des tâches qui est illustrée sur la figure 3.21. Sur cette table on trouve principalement le nom de tâche, son index et sa taille. Le "handle" est un identificateur unique sous matlab qui nous permettra de communiquer l'affectation des tâches à Matlab/Rt-lab.

	Constant4	Memory1	Integral	Fcn	Scope	Product	Display	MinMax
Constant4	0	0	2	2	0	0	0	0
Memory1	0	0	0	0	0	-1	0	0
Integral	0	0	0	0	-2	-2	0	0
Fcn	0	1	0	0	0	0	0	0
Scope	0	0	0	0	0	0	0	0
Product	0	0	0	0	0	0	0	2
Display	0	0	0	0	0	0	0	0
MinMax	0	0	0	0	0	0	2	0

Figure 3.20 : La matrice d'adjacence correspondant au modèle de la figure 3.19.

Lors de l'exploration des tâches nous vérifions la propriété "Feedthrough" des tâches afin de bien remplir la matrice d'ajacence de taille $n \times n$ de la figure 3.20. Les liens se lisent sur la matrice de la ligne vers la colonne. Si le chiffre dans la case est nul alors il n'y a pas de lien. S'il est positif, un lien direct existe entre ces deux tâches. S'il est négatif, c'est un lien "non-Feedthrough" et la valeur absolue de ce nombre donne la taille de communication. On pourra ainsi vérifier aisément la correspondance entre le modèle et les données de la matrice.

Task name	Task number	Task weight [unitless]	Task handle
Constant4	T1	1	2.002
Memory1	T2	1	3.002
Integral	T3	1	5.001
Fcn	T4	1	6.003
Scope	T5	0.1	10.001
Product	T6	1	7.004
Display	T7	0.1	8.004
MinMax	T8	1	4.002

Figure 3.21 : La table des tâches correspondant au modèle de la figure 3.19.

Finalement, pour visualiser le graphe on utilise le logiciel "GraphViz" qu'on appelle via la fonction "Representation_DAG" (voir figure 3.15), un utilitaire de représentation des graphes qui prend en entrée un fichier texte formaté selon les spécifications de GraphViz. À partir de la matrice d'adjacence et celle des attributs des tâches, une routine écrite en "script m" génère ce fichier texte et appelle "GraphViz" par la suite. On obtient le graphe tel que montré sur la figure 3.22. Sur ce graphe nous synthétisons d'une manière visuelle toute l'information de notre modèle Simulink.

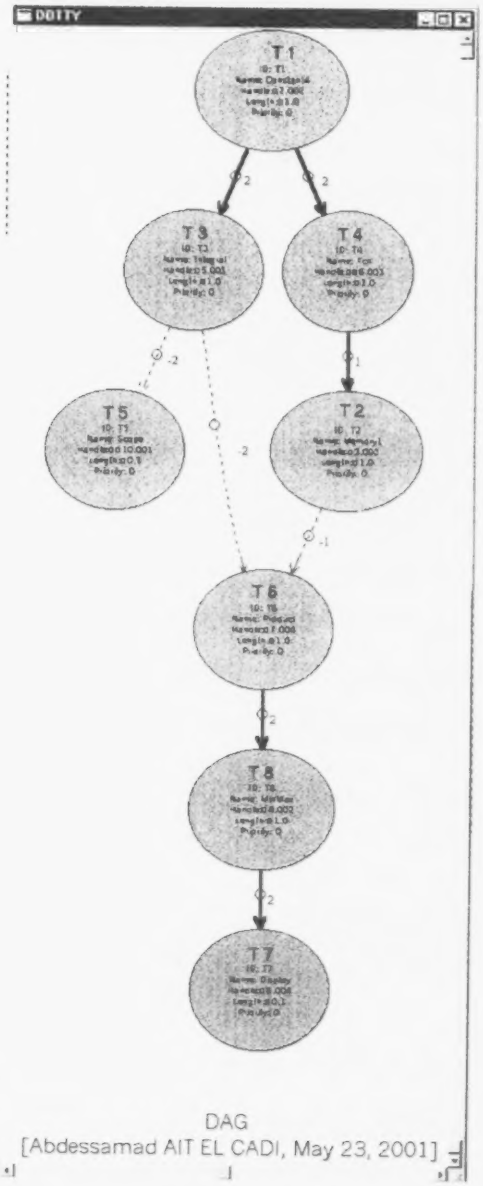

Figure 3.22 : Le DAG résultant des figures 3.19, 3.20 et 3.21.

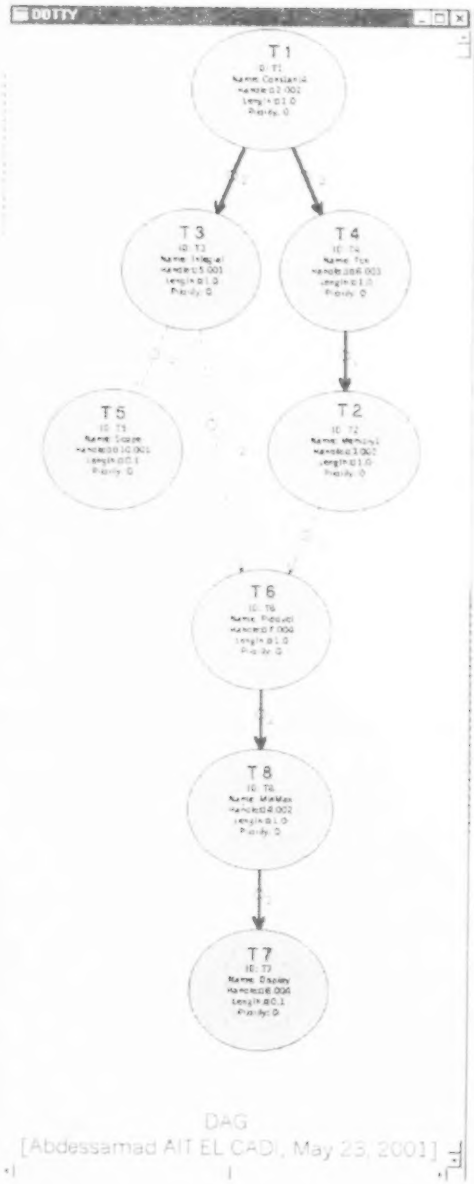

Figure 3.22 : Le DAG résultant des figures 3.19, 3.20 et 3.21.

CHAPITRE 4

RÉSOLUTION DU PROBLÈME DE PARALLÉLISATION

Dans le chapitre précédent, nous avons traité de la transformation d'un modèle Simulink en un DAG. Dans ce chapitre nous présentons la deuxième phase de notre projet, soit la parallélisation du modèle Simulink vai le DAG.

Ce chapitre 4 est composé de trois sections. La première définie le problème de parallélisation en mettant en contexte notre chapitre par rapport à l'ensemble du mémoire. La deuxième section traite de la modélisation mathématique. On présente et on compare deux types de modèles mathématiques, soit l'approche horaire et celle de séquence. On termine ce chapitre en présentant l'algorithme de résolution, soit un algorithme basé sur les règles de priorité et de "Clustering" (le "Clustering" n'est pas présenté en détails car nous n'avons fait qu'une implantation simplistique de cette stratégie).

4.1 Le problème de parallélisation

Le problème de parallélisation qu'on propose de résoudre dans ce chapitre consiste en la séparation d'un ensemble de n tâches sur m processeurs en tenant compte des contraintes de communication, de précédence et de hiérarchie. Comme il est illustré à la figure 4.1, le problème de parallélisation du DAG en haut de la figure consiste à répartir les 3 tâches sur 2 CPUs de sorte que les tâches antécédentes soient exécutées et les données retransmises avant l'exécution de toute tâche. Dans notre exemple, de deux ordinateurs identiques, nous avons le choix d'exécuter les tâches 1, 2 et 3 sur un seul CPU, les tâches 1 et 2 sur le CPU 1 et la tâche 3 sur

l'autre CPU, ou, tel qu'illustré à la figure 4.1, la tâche 1 sur le CPU 1 et les tâches 2 et 3 sur l'autre CPU. La dernière solution demande un temps total d'exécution de la durée des tâches 1 et 3 plus le temps de retransmission des données de la tâche 1 du CPu 1 au CPU 2. La solution minimisant le temps total d'exécution est retenue.

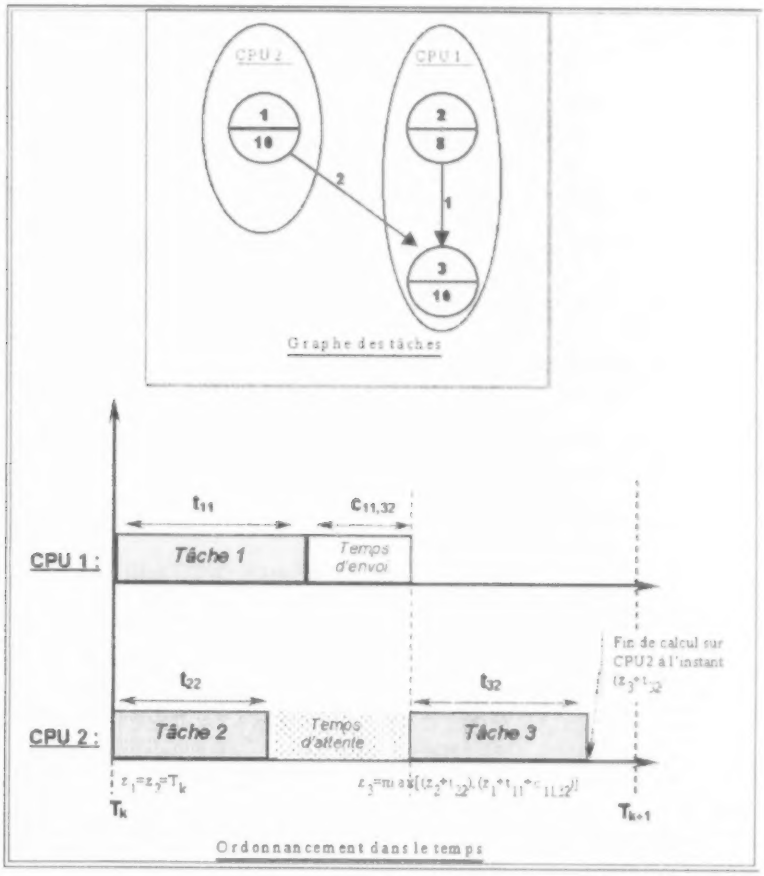

Figure 4.1 : Exemple d'ordonnancement des tâches dans le temps.

En plus, lors de la simulation d'un modèle Simulink, on doit répartir les tâches sur les différents CPU de sorte que le temps total d'exécution est plus petit qu'un

temps donné, soit le pas d'intégration $\triangle T$. On cherche ainsi à minimiser la durée totale d'exécution C_{max} (MakeSpan) tout en respectant ce pas d'intégration qui permettra la modélisation Simulink en temps réel.

Considérant N un ensemble des tâches (ensemble des indices des tâches[1]) et son graphe acyclique orienté "DAG" G=(A,N) (A : représente les arcs, N : représente les noeuds) représentant les relations d'antécédence entre les tâches; considérant M un ensemble de processeurs (ensemble des indices des processeurs[2]); le problème de parallélisation consiste à ce que chaque tâche soit affectée à un et un seul processeur et elles doivent être ordonnées en respectant les contraintes de précédence et de communication. Ce qui consiste à trouver une affectation optimale des tâches dans N aux processeurs dans M pour arriver à minimiser le temps d'exécution C_{max}.

4.2 Modélisation mathématique

Le problème de parallélisation est fort complexe. L'exercice de modéliser mathématiquement est aussi fort difficile mais il nous permet de mieux formaliser le problème en plus de nous permettre d'explorer les méthodes exactes.

Il existe deux approches à la modélisation de notre problème. La première, la plus intuitive, est celle de la confection d'un horaire. Ainsi, cette approche demande à ce que chaque tâche soit affectée à un processeur et qu'on lui donne un moment précis où cette tâche sera exécutée.

Si l'approche horaire est très intuitive, elle n'est cependant pas facile à convertir en modèle mathématique, comme on le verra plus tard. C'est pourquoi on doit faire appel à la "séquence". Ansi, d'après l'approche par séquence, un horaire de calcul

[1] $N = 1, ..., n$ *avec* n *le nombre de tâches*
[2] $M = 1, ..., m$ *avec* m *le nombre de processeurs*

parallèle peut être simplifié par l'ordonnancement (ou séquence) de l'exécution des tâches sur un processeur. Si on connaît l'affectation des tâches aux processeurs et leur séquence d'exécution, on peut retrouver l'horaire découlant. Or, la séquence des tâches est beaucoup plus facile à modifier qu'un horaire en tant que tel. C'est la raison pour laquelle l'approche par séquence est supérieure à l'approche confection d'horaire pour la modélisation mathématique et même pour sa résolution.

4.2.1 Notation

Pour faciliter la compréhension des modèles qui suivent nous définissons la notation qui sera utilisée. Soit le problème d'ordonnancement de n tâches parmi m processeurs, on définit les paramètres et les variables de décision suivantes :

*	Les données du problème :
n	Nombre de tâches
m	Nombre de processeurs (CPUs)
d_{ik}	Durée d'exécution de la tâche i sur le processeur k; $i \in N$ et $k \in M$
c_{ij}	Taille des données envoyées par la tâche i à la tâche j; $i, j \in N$
P_i	L'ensemble des prédécesseurs immédiats de la tâche i; $i \in N$
S_i	L'ensemble des successeurs immédiats de la tâche i; $i \in N$
r_{kl}	Vitesse ou taux de transmission des données entre les processeurs k et l; on a $r_{kl} = r_{lk}$; Et on pose $r_{kk} = +\infty$; avec $k, l \in M$
a_{kl}	Coût fixe de communication entre les processeurs k et l; on a $a_{kl} = a_{lk}$; Et on pose $a_{kk} = 0$; avec $k, l \in M$
ΔT	Durée d'une itération = le pas d'intégration

*	Les paramètres :
N	Ensemble des indices des tâches $N = 1, ..., n$
M	Ensemble des indices CPU $M = 1, ..., m$
C_{max}	Temps total d'exécution (MakeSpan)
E_k	Ensemble des indices des tâches affectées au processeur k; $k \in M$
$c_{il,jk}$	Durée de communication entre la tâche i sur le CPU l et la tâche j sur le CPU k; $i, j \in N$ et $l, k \in M$

*	Les variables de décision :
x_{ik}	Variable de décision pour le modèle horaire 4.2.2 $x_{ik} = 1$ si la tâche i est affectée au CPU k sinon 0; avec $i \in N$ et $k \in M$
z_i	Temps du début de l'exécution de la tâche i; $i \in N$
x^s_{ik}	Variable de décision pour le modèle par séquence 4.2.3 $x^s_{ik} = 1$ si la tâche i est affectée au CPU k à la position s sinon 0; avec $i, s \in N$ et $k \in M$

4.2.2 Modèle horaire

Le modèle suivant est inspiré des travaux de Darte (2000) et du modèle d'Urban (1998). Il a été modifié pour intégrer les contraintes de communication et être ainsi assez général. Ainsi on considère une architecture "full connected"[3] pour les processeurs et, dans le cas de mémoire partagée, on suppose la communication est instantanée. Le modèle représentant notre problème de parallélisation est le suivant :

$$Minimize \quad C_{max} \tag{4.1}$$

[3] "Full Connected" : C'est à dire que chaque ordinateur peut communiquer directement avec les autres sans intermédiaire.

$$
\begin{aligned}
&\textit{Subject to} : \\
&\sum_{k=1}^{m} x_{ik} = 1, \ \forall i \in N && (4.2)\\
&c_{il,jk} = a_{lk} + (e_{ij} \div r_{kl}), \\
&\qquad \forall (i,j) \in N, \ \forall (k,l) \in M && (4.3)\\
&z_j \geq z_i + \sum_{k=1}^{m} \sum_{l=1}^{m} [(d_{il} + c_{il,jk}\, x_{jk})\, x_{il}], \\
&\qquad \forall i \in P_j, \ \forall j \in N && (4.4)\\
&]z_i; z_i + d_{ik}[\bigcap]z_j; z_j + d_{jk}[= \emptyset, \ \forall (i,j) \in E_k, \ \forall k \in M && (4.5)\\
&C_{max} \geq z_i + d_{ik}\, x_{ik}, \ \forall i \in N, \ \forall k \in M && (4.6)\\
&C_{max} \leq \triangle T && (4.7)\\
&x_{ik} \in \{0,1\}, \ \forall i \in N, \ \forall k \in M && (4.8)\\
&z_i \geq 0, \ \forall i \in N && (4.9)
\end{aligned}
$$

La fonction objectif (4.1) minimise le temps total d'exécution C_{max}. Ce dernier est défini grâce à l'inéquation (4.6) qui détermine le temps de calcul maximal sur tous les processeurs. L'équation (4.2) traduit la contrainte d'affectation : toutes les tâches doivent être affectées. Cette contrainte permet aussi de s'assurer que chacune des tâches est bien affectée une seule fois à un seul CPU. (4.3) calcule le temps de chaque communication, ce qui peut être fait en pré-calcul. (4.4) s'occupe des contraintes de précédence. Le but de ces contraintes est de s'assurer que les tâches sont exécutées dans l'ordre défini par le DAG. Ce qui veut dire que tous les prédécesseurs i de la tâche j doivent être exécutés et leurs données transmises avant l'exécution de j. Jusque-là, rien n'interdit que deux tâches soient affectées au même processeur et soient exécutés en même temps. Il faut donc ajouter la contrainte (4.5) qui règlemente l'ordonnancement des tâches au sein du même processeur en interdisant le chevauchement des tâches au sein du même CPU. L'inéquation (4.7) est une contrainte spécifique à notre cas : elle permet de s'assurer qu'on ne dépasse

pas le pas d'intégration (la durée de l'itération de notre modèle de simulation). Finalement, les deux dernières contraintes (4.8) et (4.9) définissent la nature binaire ou continue des variables de décision.

Ce modèle n'est pas linéaire à cause des contraintes (4.4) et (4.5). Il ne peut être facilement résolu mathématiquement par des méthodes exactes.

Dans des cas particuliers, ce modèle peut être simplifié. On s'intéresse ici tout particulièrement aux cas d'architecture homogène [4] et d'architecture homogène avec mémoire partagée [5]. Ces deux cas représentent la situation de notre projet avec Opal-rt. En outre les équations sont substantiellement simplifiées et dans le deuxième cas, la contrainte (4.4) devienne linéaire. La contrainte (4.5) reste cependant complexe.

4.2.2.1 Cas d'une architecture homogène

On suppose que les processeurs sont identiques et le réseaux de communication aussi. Dans ce cas alors le temps fixe de communication et le taux de transmission sont indépendants des machines. L'équation (4.3) devient alors :

$$c_{il,jk} = a + (e_{ij} \div r) = c_{ij}, \ \forall (i,j) \in N$$

Et les durées des tâches deviennent indépendantes des machines :

$$d_{ik} = d_i \ \forall k \in M.$$

Le modèle devient :

$$Minimize \ C_{max} \qquad . \tag{4.10}$$

[4] Une architecture homogène est une architecture où toutes les composantes sont identiques. Donc les CPUs ont la même vitesse de traitement et on a le même débit de données sur tous les liens.

[5] Les CPUs partagent la même mémoire vive, l'accès aux données se fait donc dans un temps quasiment nul.

$$Subject\ to\ :$$

$$\sum_{k=1}^{m} x_{ik} = 1,\ \forall i \in N \tag{4.11}$$

$$c_{ij} = a + (e_{ij} \div r),\ \forall (i,j) \in N, \tag{4.12}$$

$$z_j \geq z_i + \sum_{k=1}^{m}\sum_{l=1}^{m} [(d_{il} + c_{ij}\, x_{jk})\, x_{il}],\ \forall i \in P_j,\ \forall j \in N \tag{4.13}$$

$$]z_i; z_i + d_i[\bigcap]z_j; z_j + d_j[= \emptyset,\ \forall (i,j) \in E_k,\ \forall k \in M \tag{4.14}$$

$$C_{max} \geq z_i + d_i\, x_{ik},\ \forall i \in N,\ \forall k \in M \tag{4.15}$$

$$C_{max} \leq \triangle T \tag{4.16}$$

$$x_{ik} \in \{0,1\},\ \forall i \in N,\ \forall k \in M \tag{4.17}$$

$$z_i \geq 0,\ \forall i \in N \tag{4.18}$$

4.2.2.2 Cas d'une architecture homogène et de mémoire partagée

Si on suppose en plus que les processeurs partagent la même mémoire, le temps de communication est négligeable et on le suppose donc nul. Le modèle devient :

$$Minimize\ C_{max} \tag{4.19}$$

$$Subject\ to\ :$$

$$\sum_{k=1}^{m} x_{ik} = 1,\ \forall i \in N \tag{4.20}$$

$$z_j \geq z_i + d_i,\ \forall i \in P_j,\ \forall j \in N \tag{4.21}$$

$$]z_i; z_i + d_i[\bigcap]z_j; z_j + d_j[= \emptyset,\ \forall (i,j) \in E_k,\ \forall k \in M \tag{4.22}$$

$$C_{max} \geq z_i + d_i.\, x_{ik},\ \forall i \in N,\ \forall k \in M \tag{4.23}$$

$$C_{max} \leq \triangle T \tag{4.24}$$

$$x_{ik} \in \{0,1\},\ \forall i \in N,\ \forall k \in M \tag{4.25}$$

$$z_i \geq 0,\ \forall i \in N \tag{4.26}$$

4.2.3 Modèle par séquence

La première formulation d'un modèle par séquence à notre problème de calcul parallèle est donné par Blazewicz (1986). Elle est basée sur la caractérisation de la position d'exécution de chaque tâche sur son CPU d'affectation. Maculan et al. (1999) a amélioré ce modèle en introduisant un troisième indice sur les variables de décisions s pour repérer la séquence des tâches. Ceci est intéressant car il permet d'éliminer l'équation (4.5). Le modèle suivant est le modèle de Maculan et al.(il ne tient pas compte des temps de communication) :

$$Minimize\ C_{max} \tag{4.27}$$

Subject to :

$$\sum_{k=1}^{m}\sum_{s=1}^{n} x_{ik}^{s} = 1,\ \forall i \in N \tag{4.28}$$

$$\sum_{i=1}^{n} x_{ik}^{1} \leq 1,\ \forall k \in M \tag{4.29}$$

$$\sum_{i=1}^{n} x_{ik}^{s} \leq \sum_{i=1}^{n} x_{ik}^{s-1},\ \forall k \in M,\ \forall s = 2, ..., n \tag{4.30}$$

$$z_j \geq z_i + \sum_{k=1}^{m}\sum_{s=1}^{n} d_{ik}\, x_{ik}^{s},\ \forall i \in P_j,\ \forall i \in N \tag{4.31}$$

$$z_j \geq z_i + d_i^k - \beta\,[2 - (x_{ik}^{s} + \sum_{r=s+1}^{n} x_{jk}^{r})],\ \forall k \in M,\ \forall s = 1, ..., n-1,\ \forall (i,j) \in N \times N \tag{4.32}$$

$$C_{max} \geq z_i + \sum_{k=1}^{m}\sum_{s=1}^{n} d_{ik}\, x_{ik}^{s},\ \forall i \in N \tag{4.33}$$

$$C_{max} \leq \Delta T \tag{4.34}$$

$$x_{ik}^{s} \in \{0,1\},\ \forall i \in N,\ \forall k \in M,\ \forall s = 1, ..., n \tag{4.35}$$

$$z_i \geq 0,\ \forall i \in N \tag{4.36}$$

La fonction objectif (4.27) est identique à (4.1), elle minimise C_{max}. L'équation (4.28) s'assure que chaque tâche est affectée à un seul CPU. Les inéquations (4.29)

et (4.30) évitent le chevauchement des tâches au sein d'un même CPU, ce qui est équivalent à (4.5). Comme l'inéquation (4.4) du modèle horaire, l'inéquation (4.31) s'occupe de faire respecter la contrainte de précédence. L'inéquation (4.32) définit le temps de début de chaque tâche. Dans cette inéquation β est une constante contenant un grand nombre. Puis, (4.33) tout comme (4.6), calcule le C_{max}. (4.34) fait respecter le pas de simulation identiquement à (4.7). Finalement, les équations (4.35) et (4.36) définissent la nature binaire ou continue des variables de décision.

Le modèle par séquence est linéaire, une amélioration par rapport au modèle horaire. Cependant, le modèle par séquence requière plus de variables que le modèle horaire en raison de l'ajout d'un troisième indice s aux variables x_{ik}^{s} pour tenir compte de la séquence. Le nombre de variables requis par le modèle horaire, un nombre augmentant exponentiellement avec le nombre de tâches dans le DAG, est déjà énorme. Ainsi pour un problème de 100 tâches à séparer sur 10 CPUs, on aura approximativement pour le modèle horaire général décrit à la section 4.2.2 :

- 1 100 variables (de l'ordre de $n\,m$) [6];
- 1 112 201 contraintes (de l'ordre de $n^2\,m^2$) [7];

Puisque le modèle par séquence requière de nombreuses variables binaires, même un petit problème d'ordonnancement ne peut pas être résolu sur un logiciel commercial de programmation linéaire tel CPLEX dans un temps raisonable. C'est pourquoi la communauté scientifique travaille sur la formulation et la résolution de cas particuliers, espérant faire des progrès vers le problème initial, ainsi que sur le développement de méthodes heuristiques et métaheuristiques pour résoudre les problèmes de taille réelle.

[6] Pour le modèle par séquence le nombre de variables est de l'ordre de $n^2.m$

[7] Sans communication, dans le modèle horaire le nombre de contraintes est de l'ordre de $n^2.m$. Alors que pour le modèle par séquence -sans communication- le nombre de contraintes est de l'ordre de $n^3.m$

Ainsi, dans la suite de ce chapitre, la résolution de ce problème se fera par le biais d'heuristiques.

4.3 Algorithme de résolution

Dans cette section, nous présentons en détail l'heuristique de résolution du problème de parallélisation basée sur la règle de priorité. Cette heuristique prend un modèle Simulink schématisé sous forme de DAG et propose des solutions d'ordonnancement. Avant de présenter les détails de l'algorithme implanté en "script m"[8], nous débutons par la description de la logique de cette méthode.

4.3.1 Logique des heuristiques basées sur la règle de priorité

L'heuristique basée sur la règle de priorité suggère, à partir des données d'un DAG, un ordonnancement faisable (et de bonne qualité) des tâches définies par le DAG sur les CPUs disponibles. Comme son nom l'indique, elle commence par un calcul d'une priorité à associer à chaque tâche. Les tâches sont classées selon cette priorité. Puis on commence à affecter les tâches une à une aux CPUs disponibles tout en respectant les contraintes de précédence et la priorité. On a terminé lorsque toutes les tâches sont affectées aux CPUs.

La procédure peut être résumée comme suit :

1. Calculer la priorité à associer à chaque tâche;

2. Classer les tâches selon cette priorité;

[8] "script m" est un script utilisé par Matlab. C'est le contexte de travail qui a imposé que le code soit fait en ce langage

3. Identifier les tâches disponibles (une tâche disponible est une tâche qui n'a pas de prédécesseurs ou tous les prédécesseurs ont déjà été affectés);

4. Sélectionner la tâche disponible la plus prioritaire;

5. Affecter cette dernière au CPU minimisant le temps d'achèvement de cette tâche (Ceci est une stratégie que nous avons appelé "MinMin". Sinon, on peut choisir le CPU où la tâche se termine le plus tard - stratégie "MinMax"- ou choisir le CPU d'une manière aléatoire - stratégie "Ala";

6. Mettre à jour les données (surtout ce qui est relatif aux tâches disponibles);

7. Si toutes les tâches sont affectées, arrêter, sinon aller à l'étape 3.

Pour le calcul des priorités, on peut procéder de différents façons. En fait, on utilise plusieurs règles de priorité puisque cela nous permet de générer plusieurs solutions différentes et ainsi choisir la meilleure. Les différentes types de priorité que nous calculons sont :

1. Aléatoire : On affecte à chaque tâches une priorité tirée d'une manière aléatoire;

2. Le plus long chemin : Pour chaque tâche on calcule le chemin le plus long à partir de la tâche jusqu'à la fin du graphe. On utilise des algorithmes comme "Dijkstra". Voir El-Rewini et al. (1994) pour un exemple d'implantation dans le cadre d'un problème de parallélisation.

3. Niveau dans le graphe;

4. Nombre de successeurs (tous ou juste les immédiats);

5. Quantité de calcul dépendante de la tâche (somme des durées des successeurs);

6. La durée de la tâche elle-même, la plus grande tâche en premier.

Chacune de ces règles de priorité, sauf celle aléatoire, essait d'identifier les tâches au début du graphe afin de les affecter en premier. La règle de la tâche la plus grande, quant à elle, met l'emphase sur l'affectation des tâches les plus grandes en tout premier lieu.

4.3.2 Exemple d'application

Soit le modèle Simulink "test-heuristique" de la figure 4.2 dont le DAG est donné à la figure 4.3 (il s'agit simplement d'un exemple d'illustration, sans aucune fonctionnalité). On cherche à le séparer sur deux CPUs. L'exécution de notre algorithme de génération du DAG nous donne la matrice d'adjacence et la tables des durées des tâches. Ces deux tables, montrées aux figures 4.4 et 4.5, sont l'entrée de départ de notre algorithme.

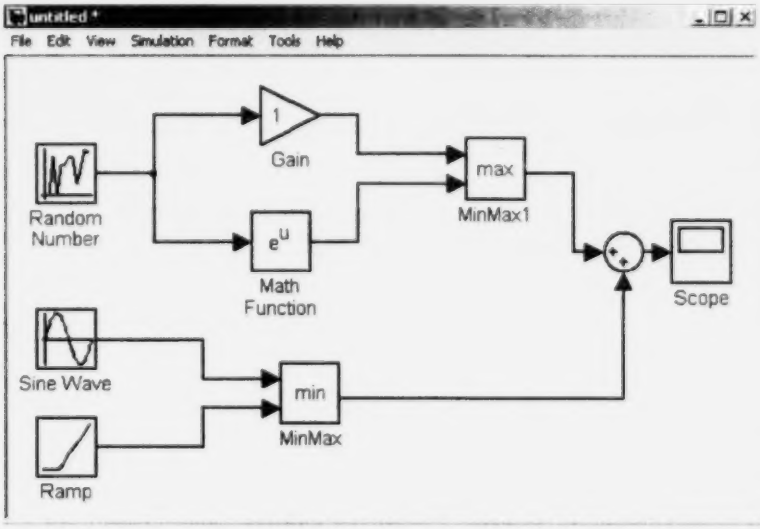

Figure 4.2 : Modèle Simulink "test-heuristique".

Premièrement, on détermine la priorité des tâches. Dans cet exemple on utilise le

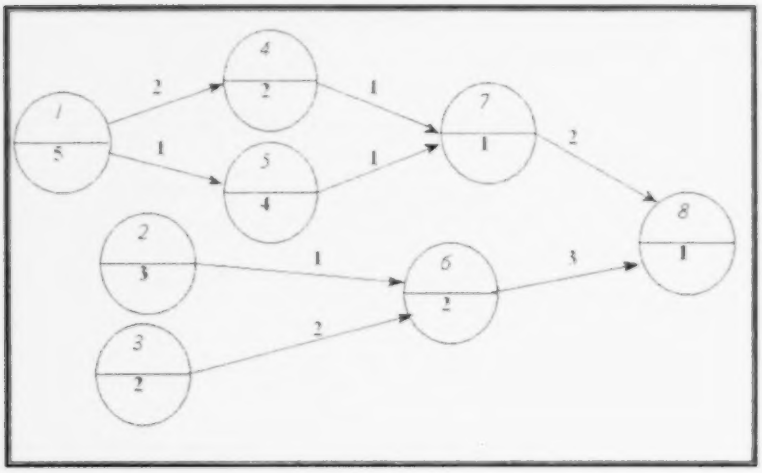

Figure 4.3 : DAG du modèle Simulink de la figure 4.2.

Links	1	2	3	4	5	6	7	8
1				2	1			
2						1		
3						2		
4							1	
5							1	
6								3
7								2
8								

Figure 4.4 : Matrice d'adjacence issue du DAG de la figure 4.2.

Task	1	2	3	4	5	6	7	8
Length	5	3	2	2	4	2	1	1

Figure 4.5 : Liste des tâches avec leurs durées issue du DAG de la figure 4.2.

chemin le plus long qui est calculé par l'algorithme de "Dijkstra" (El-rewini et al., 1994). On mesure ici la durée de la distance entre le début d'une tâche donnée et la fin du graphe. La tâche avec le chemin le plus long est considérée comme étant la plus prioritaire. La figure 4.6 donne le résultat du calcul de la priorité.

Task	1	2	3	4	5	6	7	8
Priority	15	10	10	7	9	6	4	1
End at								

Figure 4.6 : Liste des tâches avec leurs priorités respectives calculées par l'algorithme de "Dijkstra" à partir du DAG de la figure 4.2.

Deuxièmement, on trie les tâches selon leur priorité. La figure 4.7 montre les tâches et leur priorité une fois cette dernière triée selon l'ordre décroissant.

Task	1	2	3	5	4	6	7	8
Priority	15	10	10	9	7	6	4	1
End at								

Figure 4.7 : Liste des tâches avec leurs priorités triées selon un ordre décroissant.

Troisièmement, on repère les tâches disponibles (tâches qui n'ont pas de prédécesseurs

ou dont tous les prédécesseurs sont déjà affectés à un CPU). D'après le DAG de la figure 4.3 les tâches : 1; 2 et 3 n'ont pas de prédécesseurs (voir figure 4.8). Parmi ces trois tâches, la plus prioritaire est la tâche 1, on la sélectionne.

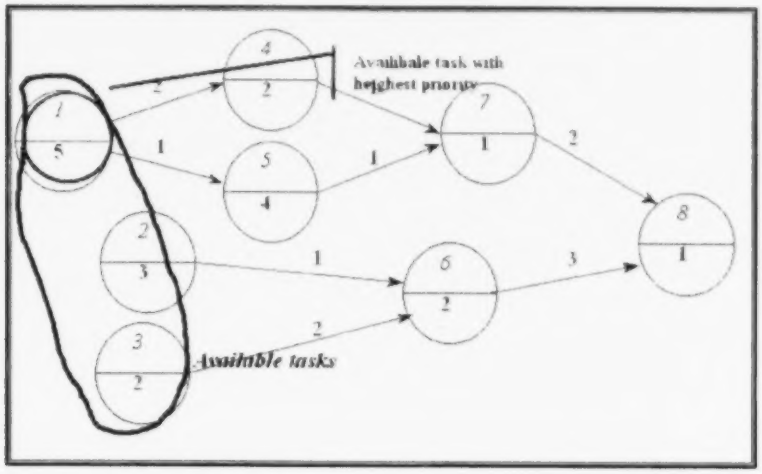

Figure 4.8 : Les tâches disponibles avant le début de l'affectation.

Quatrièmement, on affecte cette tâche au CPU où elle peut commencer le plutôt possible, c'est-à-dire le CPU 1 comme indiqué à la figure 4.9.

Cinquièmement, on met à jour les données, surtout l'ensemble des tâches disponibles, car il y a des tâches qui peuvent devenir disponibles suite à l'affectation d'une tâche. L'ensemble des tâches disponibles devient 4; 5; 2 et 3 (voir figure 4.10).

Sixièmement, vu que toutes les tâches ne sont pas encore affectées, on refait les étapes 2 à 5 jusqu'à l'effectation de toutes les tâches.

La tâche suivante à affecter est la tâche 2. On l'affecte au CPU 2 car c'est lui qui permet de la compléter le plutôt possible. L'ensemble des tâches disponibles devient : 4; 5 et 3. La tâche la plus prioritaire de cet ensemble est la tâche 3.

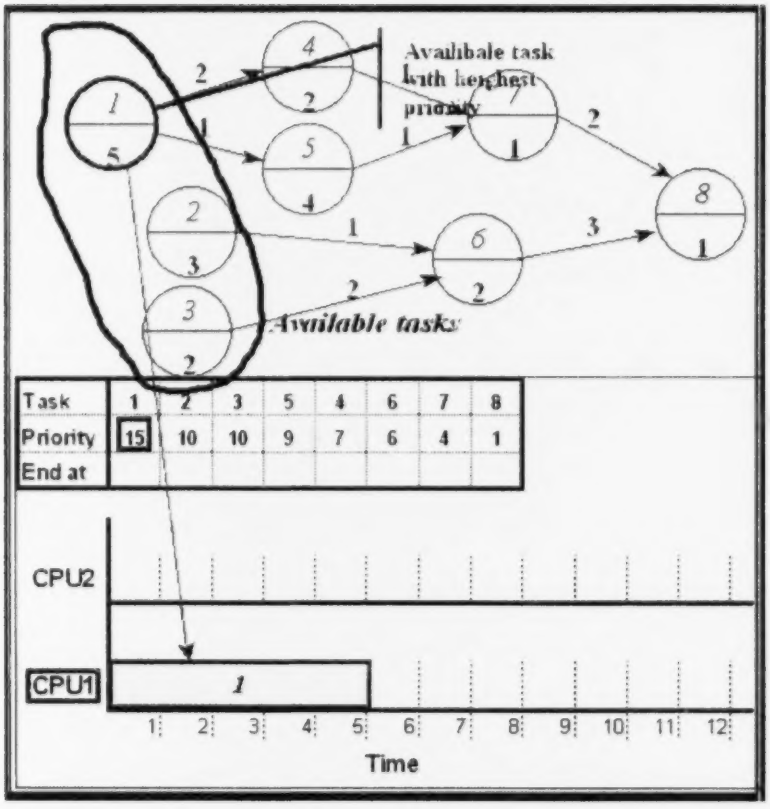

Figure 4.9 : Affectation de la tâche disponible “1” au CPU 1.

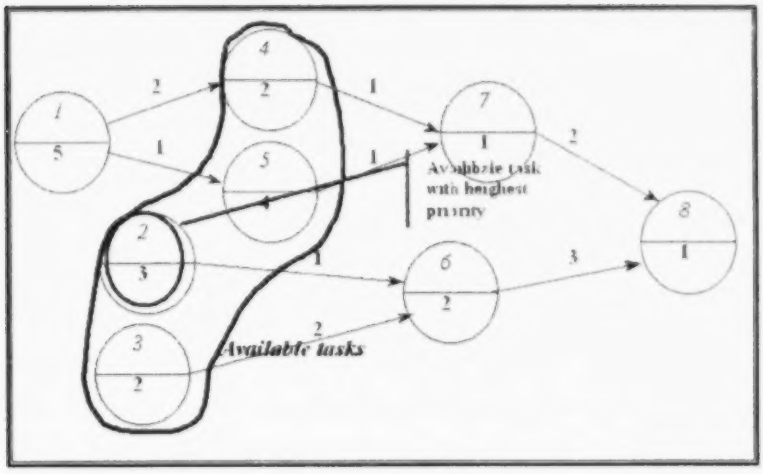

Figure 4.10 : Les tâches disponibles après la première affectation.

On l'affecte à son tour au CPU 2. L'affectation et l'horaire final est donné sur le diagramme de Gantt de la figure 4.11. On peut extraire de ce diagramme les mesures de performance suivantes :

- la durée totale de l'exécution du modèle : $C_{max} = 12$;
- le pourcentage d'utilisation du CPU 1 : $= 9 \div 12 = 75\%$;
- le pourcentage d'utilisation du CPU 2 : $= 11 \div 12 = 91.6\%$;
- le facteur d'accélération : $SpeedUp = 20 \div 12 = 1.66$.

4.3.3 L'algorithme

Notre méthode de résolution est maintenant présentée dans sa forme algorithmique. L'algorithme d'ordonnancement développé ici est basé sur l'heuristique décrite par El-Rewini et al. (1998). Il a été modifié et amélioré pour correspondre à notre

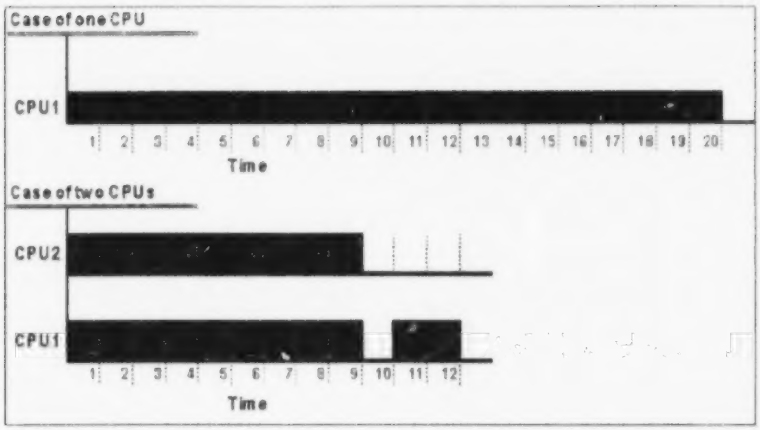

Figure 4.11 : Résultat de la séparation du modèle Simulink de la figure 4.2 sur deux CPUs.

cas. Principalement, l'algorithme prend en entrée un DAG comme sur la figure 4.3 et suggère différents ordonnancements comme celui de la figure 4.11. Le DAG est représenté par une matrice d'adjacence (figure 4.4) et par une une table d'attributs des tâches (figure 4.5). Nous utilisons cette représentation, au lieu de la liste d'adjacence, car on travaille sous Matlab qui est un logiciel matriciel et qui traite très bien les matrices creuses.

4.3.3.1 Programme principal

Le programme sous Matlab est un script contenant des instructions et pouvant lui-même appeller d'autres sous-scripts pour l'exécution d'autres fonctions. La fonction principale est :

RWT(Adjacency,TaskLength).

La fonction prend la matrice d'adjacence "Ajacency" et la table de définition des tâches "TaskLength", générées par le générateur de DAG, et retourne les sept

solutions calculées tout en indiquant laquelle est la meilleure. Le script RWT.m, de cette fonction, est présenté ci-dessous, suivi d'un texte explicatif. Cependant les sous-fonctions ne sont pas présentées car leur détail n'est pas nécessaire à la compréhension de l'algorithme.

```
function [Solution, Best] = RWT(Adjacency, TaskLength)
1. globalComPrm;
2. global NbTask;
3. if nargin==0
4.       [Adjacency] = CreateAdjacencyMatrix;
5.       TaskLength = ones(length(Adjacency),1);
6. elseif nargin==1
7.       TaskLength = ones(length(Adjacency),1)
9. elseif nargin > 2
10.      disp('This function should have only two, one or no
                arguments');
11. end;
12. NbTask = length(A);
13. Priority(1) = {'Dijkstra'};
14. Priority(2) = {'GraphLevel'};
15. Priority(3) = {'TaskLength'};
16. Priority(4) = {'Random'};
17. Priority(5) = {'SumChild'};
18. Priority(6) = {'NbChild'};
19. for i=1:6
20.      [Task, CPU, Adjacency] = Initialization(Adjacency,
                                                 TaskLength);
21.      [Task] = CalculatePriority(Adjacency, Task, Priority{i});
22.      [AvailableTask] = GetNextAvailableTask(Adjacency, Task);
```

```
23.     while not(isempty(AvailableTask))
24.             [AvailableCPU, When, Position] =
                GetAvailableCPU(Adjacency,Task,AvailableTask,CPU);
25.             [Task, CPU] = AssignIt(Task, AvailableTask, CPU,
                                AvailableCPU, When, Position);
26.             AvailableTask =GetNextAvailableTask(Adjacency,Task);
27.     end;
28.     Solution(i+1).Type = Priority(i);
29.     [Solution(i).Task, Solution(i).CPU] = CorrectFormatSol(Task,
                                                    CPU, NbTask);
30.     [Solution(i+1).StepSize] = GetStepSize(CPU);
31. end;
32. [Task, CPU,Adjacency] = Initialization(Adjacency,TaskLength);
33. [SubGraph] = GetSubGraph(Adjacency, Task);
34. [SolutionSubGraph] =  BalanceSubGraph(SubGraph, CPU);
35. [Solution(7).Task, Solution(7).CPU] =
            CorrectFormatSol(SolutionSubGraph.Task,
                                SolutionSubGraph.CPU, NbTask);
36. Solution(7).Type = {'BySubGraph'};
37. [Solution(7).StepSize] = GetStepSize(Solution(7).CPU);
38. [Best] = GetBestSolution(Solution);
```

Les lignes 1 et 2 déclarent les variables globales partagées par tous les scripts .m. La variable "ComPrm" contient les informations sur les délais de communication et permet deux modélisations soit "temps constant" soit "temps affine". La variable "NbTask" contient le nombre de tâches dans le DAG. Les lignes 3 à 11 vérifient la cohérence des arguments d'entrée. Si aucun argument n'est donné en entrée, le

script "CreateAdjacencyMatrix.m" est appelé afin de générer un exemple pour des fins de test. Si le nombre d'argument et trop grand, on affiche un message d'erreur invitant l'utilisateur à corriger sa saisie. La Ligne 12 retourne le nombre de tâches dans le DAG. Les lignes 13 à 18 identifient les six règles de priorité utilisées par le programme, soit :

Dijkstra Utilise le chemin le plus long à partir de la tâche jusqu'à la fin du DAG;

GraphLevel Utilise le niveau dans le graphe : nombre de tâches de la tâches jusqu'à la fin du DAG;

TaskLength Utilise la durée de la tâche : la plus longue est la plus prioritaire;

Random Utilise une priorité tirée de manière aléatoire;

SumChild Utilise la somme des durées des tâches suivantes;

NbChild Utilise le nombre des tâches suivantes.

Les lignes 19 à 31 contiennent l'heuristique basée sur la règle de priorité. L'heuristique est exécutée six fois pour les six différentes priorités. Les lignes 32 à 37 construisent la septième solution à partir d'un concept complètement différent de la règle de priorité. Il est basée sur du "Clustering" regroupant les tâches sur un algorithme du type "Glouton" dans la matrice "SubGraph". Finalement, la ligne 38 compare toutes les sept solution pour sélectionner la meilleure.

CHAPITRE 5

TESTS ET ANALYSE DES RÉSULTATS

Ce dernier chapitre est consacré à l'ensemble des tests utilisés pour valider notre approche. Nous réalisons ces test en trois grandes phases qui correspondent aux trois sections de ce chapitre. La première phase concerne des tests d'ordre général pour mesurer la performance et le comportment de notre heuristique d'ordonnancement. La deuxième phase permet de tester l'ensemble du projet, depuis l'analyse du modèle Simulink à la création du modèle séparé et son exécution en passant par la génération du DAG. La dernière phase teste la robustesse de notre solution face à un ordonnanceur qu'on ne contrôle pas totalement.

Ainsi, dans la première section de ce chapitre on définit un protocole de test général basé sur un ensemble de graphes diversifiés. Le but de cette section est de mesurer la performance des différentes stratégies et ainsi en déterminer la plus appropriée selon la nature du graphe.

Dans la deuxième section, on présente les résultats d'un test pratique sur un modèle Simulink de deux robots. Le test porte sur l'ensemble du module à automatiser et le but est de vérifier la qualité des solutions en terme de respect de l'intégrité du modèle original et de la performance obtenue avec notre heuristique.

Finalement, la troisième section reprend les solutions de la première section pour réaliser un test de robustesse sur un ordonnanceur qu'on ne contrôle pas totalement. En effet, on ne peut pas, présentement, imposer la séquence d'exécution des tâches sur chaque CPU avec la version actuelle de Rt-lab : on peut seulement définir l'ensemble des tâches exécutées par chaque CPU. Il y a donc un risque de décalage

entre l'horaire prédit par l'algorithme et l'horaire réel exécuté dans l'environnement Rt-lab. Ainsi le but de ces tests est de mesurer ce décalage et de trouver les meilleures affectations dans cet environnement.

5.1 Tests de performance

Le premier jeu de tests nous permet d'évaluer la performance de notre heuristique et de déterminer ses limites. Il s'agit de tester l'algorithme d'ordonnacement dans un environnement contrôlé en utilisant différentes règles de priorité et différentes stratégies d'affectation. Les tests sont réalisés sur un ensemble varié de graphes élaborés par Scholl (1999) et qui sont disponible sur le site web :

http://www.bwl.tu-darmstadt.de/bwl3/forsch/projekte/alb/albdata.htm

5.1.1 Objectif du test

Le but de ces tests est d'évaluer notre heuristique, trouver les meilleures stratégies d'affectation et les meilleures règles de priorité pour chaque type de graphes et propriétés des communications. Cet exercice est fort utile car il nous aidera à mieux comprendre les résultats de nos tests pratiques effectués dans un contexte beaucoup plus complexe.

5.1.2 Les données pour le test

Les données principales du test proviennent de Scholl (1999). Ce jeu de graphes a été défini pour tester des algorithmes d'équilibrage d'une ligne d'assemblage. Ce sont des graphes de tâches où les liens de précédence ne contiennent aucun poids (pas de communication). Ils sont toutefois bien choisis et couvrent une large gamme de graphes selon la densité du graphe, le nombre de tâches, degré de convergence

et de divergence. Le tableau 5.1 donne une description sommaire de ces graphes :

- Sur la première colonne on a le nom du fichier de données.
- Sur la deuxième colonne on a le nom du graphe tel que baptisé par Scholl.
- Sur la troisième colonne on a "n", le nombe de tâches.
- Sur la quatrième, la cinquième et la sixième colonne on lit respectivement : "tmin" la durée de la plus petite tâche, "tmax" de la plus grande tâche et "tsum" la somme des durées.
- Sur la septième colonne on a "OS" (order strength [=number of all precedence relations $/(n*(n-1))$]), la force du degré du graphe.
- Sur la huitième colonne on a "TV" (time variability ratio [= $tmax/tmin$]) ratio de variabilité des durées des tâches.
- Sur la neuvième et la dixième colonne, on a respectivement la divergence du graphe "div" et sa convergence "conv". La divergence du graphe est définie par :

  $$div(G) = 1 - \sum_{j \in N \setminus \{r\}} (d_j - 1) \div Card(A)$$

  où N est l'ensemble des noeuds du graphe G, A l'ensemble des arcs, j un noeud quelconque, d_j degré entrant de j et r est le noeud d'entrée du graphe (s'il y'en a plusieurs il faudra ajouter un noeud fictif qui sera l'antécédent de ces derniers et, par la suite, l'unique entrée du graphe G). La convergence est définie de la même façon en considérant d_j le degré sortant de j et r est l'unique noeud de fin du graphe G (voir Scholl (1999) pour plus de détails). Ainsi, plus un graphe contient des noeuds avec plusieurs arcs sortant, plus il sera "divergeant", et vice versa.
- Sur la dernière colonne on a le degré moyen du graphe, qui est le nombre moyen d'arcs par noeud.

Tableau 5.1 : Description du jeu de graphes utilisés pour le test de performance.

File	Name	n	tmin	tmax	tsum	OS	TV	div	conv	d Moyen
ARC83	Arcus1	83	233	3691	75707	59,09	15,84	0,73	0,73	1,61
ARC111	Arcus2	111	10	5689	150399	40,38	568,9	0,63	0,63	1,36
BARTHOLD	Barthold	148	3	383	5634	25,8	127,67	0,74	0,72	1,18
BARTHOL2	Barthol2	148	1	83	4234	25,8	83	0,74	0,72	1,18
BOWMAN8	Bowman	8	3	17	75	75	5,67	0,88	0,8	1,00
BUXEY	Buxey	29	1	25	324	50,74	25	0,74	0,78	1,24
GUNTHER	Gunther	35	1	40	483	59,5	40	0,78	0,76	1,29
HAHN	Hahn	53	40	1775	14026	83,82	44,38	0,63	0,63	1,55
HESKIA	Heskiaoff	28	1	108	1024	22,49	108	0,68	0,69	1,39
JACKSON	Jackson	11	1	7	46	58,18	7	0,77	0,77	1,18
JAESCHKE	Jaeschke	9	1	6	37	83,33	6	0,73	0,73	1,22
KILBRID	Kilbridge	45	3	55	552	44,55	18,33	0,67	0,69	1,38
LUTZ1	Lutz1	32	100	1400	14140	83,47	14	0,76	0,82	1,19
LUTZ2	Lutz2	89	1	10	485	77,55	10	0,75	0,75	1,33
LUTZ3	Lutz3	89	1	74	1644	77,55	74	0,75	0,75	1,33
MANSOOR	Mansoor	11	2	45	185	60	22,5	0,79	0,91	1,00
MERTENS	Mertens	7	1	6	29	52,38	6	1	0,78	0,86
MITCHELL	Mitchell	21	1	13	105	70,95	13	0,74	0,7	1,29
MUKHERJE	Mukherje	94	8	171	4208	44,8	21,38	0,51	0,51	1,93
ROSZIEG	Roszieg	25	1	13	125	71,67	13	0,74	0,69	1,28
SAWYER30	Sawyer	30	1	25	324	44,83	25	0,83	0,83	1,07
TONGE70	Tonge	70	1	156	3510	59,42	156	0,78	0,73	1,23
WARNECKE	Warnecke	58	7	53	1548	59,1	7,57	0,72	0,81	1,21
WEE-MAG	Wee-Mag	75	2	27	1499	22,67	13,5	0,85	0,62	1,16

Pour pouvoir utiliser ces graphes dans notre contexte où la durée de communication est importante, nous avons défini une durée de communication sur chaque arc. Pour chaque graphe générique, nous avons généré trois graphes avec les contextes de communication suivants[1] :

À communication Équivalente la durée de communication est équivalente à la durée de la tâche moyenne (la durée de communication moyenne = la durée de tâche moyenne).

À communication Nulle la durée de communication est négligeable devant la durée de la tâche moyenne (la durée de communication moyenne = 0.01∗ la durée de tâche moyenne).

À communication Prépondérante la durée de communication est très grande devant la durée de la tâche moyenne (la durée de communication moyenne = 100∗ la durée de tâche moyenne).

Ce jeux de données est disponible sur le site du projet :

http://www.crt.umontreal.ca/~aitelcad/Projets/Parallelisation/

5.1.3 Le protocole du test

Le test consiste en l'application de notre programme "RWT" (Ranked Weigthed Tasks) à chacun des graphes des trois contextes de communication pour résoudre le problème d'exécution des tâches sur deux CPU.

[1]Ces ordres de grandeurs pour les durées de communication (0.01; 1; 100) sont pris pour traduire les trois grand types de situations chez Opal-rt.

5.1.3.1 Les différents scénarios à tester

Pour chaque graphe, nous exécutons notre heuristique avec six règles de priorité mais également avec trois superbes règles d'affectation aux CPUs. Elles sont les suivantes :

- Règles d'affectation des tâches aux CPUs :
  1. Optimiste "Min-Min" : Nous affectons la tâche au CPU où elle commence le plus tôt.
  2. Pessimiste "Min-Max" : Nous affectons la tâche au CPU où elle commence le plus tard.
  3. Aléatoire "Aléa" : Le choix du CPU est aléatoire.
- Les six règles de priorité :
  1. Solution 1 : utilisant le chemin le plus long "Dijkstra".
  2. Solution 2 : utilisant le niveau dans le graphe "GraphLevel".
  3. Solution 3 : utilisant la durée de la tâche elle-même "TaskLength".
  4. Solution 4 : utilisant une priorité aléatoire "Random".
  5. Solution 5 : utilisant la somme des durées des enfants "SumChild".
  6. Solution 6 : utilisant le nombre d'enfants (successeurs immédiats) "NbChild".

5.1.3.2 Les données à collecter et les mesures de performance

Les données à collecter à la fin du test sont :

- Le nom du graphe utilisé.
- Le type de communication (Nulle, Équivalente ou Prépondérante).

- Le nom de la stratégie d'affectation.
- Le nom de la régle de priorité.
- Le C_{max} réalisé (le makespan).

De même nous calculons les indicateurs suivants :

- Le pourcentage d'utilisation de chacun des CPUs.
- Le facteur d'accélération "SpeedUp":

$$SpeedUp = t_{sum}/C_{max}.$$

Pour le bon fonctionnement des tests un "TestManager" à été programmé pour exécuter ces différents scénarios. À la fin nous générons le fichier "ASCII" (disponible sur le site du projet : *http://www.crt.umontreal.ca/~aitelcad/Projets/Parallelisation/*) où chaque ligne contient les informations décrite dans le tableau 5.2.

Tableau 5.2 : Données contenu dans le fichier résultat du TestManager

Type de communication	Stratégie	Graphe	Durée de résolution	Les résultats sur la même ligne pour chaque régle de priorité une après l'autre

5.1.4 Les résultats du test

Comme indiqué dans le protocole du test, nous avons réccupéré pour chaque graphe les cinquante-quatre solutions (trois stratégies, six règles de priorités et trois types de communication). Les résultats des 24 graphes sont résumés sous forme de trois tableaux : le tableau 5.3 donne les facteurs d'accélération moyens "SpeedUp"; les tableaux 5.4 et 5.5 contiennent respectivement le pourcentage d'utilisation moyen

du CPU 1 et du CPU 2. Finalement, le tableau 5.6 donne le facteur d'accélération moyen - sur deux CPUs - pour chaque graphe-test et règle de priorité.

Tableau 5.3 : Facteur d'accélération "SpeedUp" selon le type de communication, la stratégie d'affectation et la règle de priorité pour les graphes-tests

Type de Communication	Stratégie	Règle de priorité					
		Dijkstra	GraphLevel	TaskLength	Random	SumChild	NbChild
Equivalente	Alea	1,11	1,00	0,98	0,97	0,96	1,04
	MinMax	0,83	0,79	0,74	0,74	0,76	0,79
	MinMin	1,52	1,44	1,42	1,42	1,44	1,46
Nulle	Alea	1,41	1,31	1,32	1,27	1,35	1,38
	MinMax	1,09	1,15	1,13	1,13	1,11	1,07
	MinMin	1,74	1,62	1,63	1,58	1,64	1,67
Prépondérante	Alea	0,04	0,03	0,04	0,04	0,03	0,03
	MinMax	0,02	0,02	0,02	0,02	0,02	0,02
	MinMin	0,54	0,54	0,56	0,54	0,54	0,54
	Moyenne	0,92	0,88	0,87	0,86	0,87	0,89

5.1.5 Analyse des résultats.

D'après l'analyse des tableaux 5.3 à 5.6, quatre résultats importants émergent :

(1) La stratégie "MinMin" est toujours la meilleure : elle dépasse de loin les deux autres stratégies.

(2) Pour les règles de priorité, on constate que, quelque soit la stratégie et le type de communication, la règle du chemin le plus long "Dijkstra" est celle qui donne en moyenne les meilleurs résultats.

(3) Quand la communication est très prépondérante (c.-à-d. les temps de communication sont plus grands que le temps des tâches), tous les "SpeedUp" sont inféreurs à 1. La parallélisation est donc inintéressante. Ceci peut être dû à une inefficacité de notre heuristique mais il est plutôt probable que c'est intrinsèque au graphe en tant que tel.

Tableau 5.4 : Pourcentage d'utilisation du CPU 1 selon le type de communication, la stratégie d'affectation et la règle de priorité pour les graphes-tests

Type de Communication	Stratégie	Règle de priorité					
		Dijkstra	GraphLevel	TaskLength	Random	SumChild	NbChild
Equivalente	Alea	50,1%	51,5%	46,9%	51,4%	47,8%	51,1%
	MinMax	43,2%	40,7%	37,0%	36,2%	39,5%	39,5%
	MinMin	86,3%	80,5%	74,8%	72,0%	79,7%	80,0%
Nulle	Alea	77,9%	65,8%	69,2%	65,2%	65,1%	70,8%
	MinMax	55,9%	58,0%	59,9%	57,9%	59,0%	63,9%
	MinMin	94,7%	83,5%	84,6%	77,9%	88,0%	83,0%
Prépondérante	Alea	1,8%	1,5%	2,0%	1,8%	1,6%	1,9%
	MinMax	1,1%	1,1%	1,1%	1,1%	1,1%	1,1%
	MinMin	50,8%	48,3%	44,0%	45,6%	46,5%	47,8%
	Moyenne	51,3%	47,9%	46,6%	45,5%	47,6%	48,8%

Tableau 5.5 : Pourcentage d'utilisation du CPU 2 selon le type de communication, la stratégie d'affectation et la règle de priorité pour les graphes-tests

Type de Communication	Stratégie	Règle de priorité					
		Dijkstra	GraphLevel	TaskLength	Random	SumChild	NbChild
Equivalente	Alea	60,5%	48,2%	50,8%	45,6%	48,2%	53,1%
	MinMax	39,6%	38,7%	37,3%	37,4%	36,1%	39,7%
	MinMin	66,1%	63,7%	66,9%	69,6%	64,2%	65,6%
Nulle	Alea	63,3%	65,4%	62,6%	62,1%	69,5%	67,0%
	MinMax	52,9%	57,2%	53,6%	55,3%	52,3%	42,7%
	MinMin	79,6%	78,3%	78,0%	80,6%	75,9%	84,0%
Prépondérante	Alea	1,8%	1,8%	1,9%	1,8%	1,6%	1,6%
	MinMax	1,0%	1,0%	1,0%	1,0%	1,0%	1,0%
	MinMin	3,0%	5,4%	12,0%	8,3%	7,4%	6,1%
	Moyenne	40,9%	40,0%	40,4%	40,2%	39,6%	40,1%

Tableau 5.6 : Facteur d'accélération moyen "SpeedUp" pour chaque graphe-test et règle de priorité.

	Solution selon la régle de priorité					
Graphe	Dijkstra	GraphLevel	TaskLength	Random	SumChild	NbChild
ARC111	1,91	1,64	1,67	1,57	1,76	1,74
ARC83	1,73	1,57	1,67	1,53	1,68	1,61
BARTHOL2	1,98	1,92	1,94	1,87	1,97	1,98
BARTHOLD	2,00	1,91	1,83	1,81	1,98	1,99
BOWMAN8	1,36	1,36	1,36	1,36	1,36	1,36
BUXEY	1,85	1,64	1,68	1,66	1,56	1,84
GUNTHER	1,83	1,69	1,71	1,51	1,71	1,69
HAHN	1,39	1,18	1,35	1,21	1,22	1,20
HESKIA	1,85	1,79	1,86	1,77	1,61	1,69
JACKSON	1,53	1,48	1,53	1,53	1,48	1,58
JAESCHKE	1,32	1,32	1,32	1,32	1,27	1,27
KILBRID	1,91	1,78	1,57	1,73	1,70	1,86
LUTZ1	1,67	1,56	1,50	1,46	1,58	1,55
LUTZ2	1,90	1,77	1,63	1,60	1,61	1,79
LUTZ3	1,53	1,47	1,51	1,39	1,46	1,46
MANSOOR	1,58	1,31	1,50	1,59	1,55	1,58
MERTENS	1,71	1,70	1,71	1,70	1,71	1,71
MITCHELL	1,42	1,41	1,38	1,38	1,38	1,38
MUKHERJE	1,79	1,67	1,68	1,67	1,73	1,69
ROSZIEG	1,67	1,64	1,56	1,62	1,64	1,69
SAWYER30	2,00	1,80	1,85	1,59	1,82	1,90
TONGE70	1,98	1,64	1,74	1,77	1,95	1,78
WARNECKE	1,97	1,70	1,61	1,54	1,66	1,79
WEE-MAG	1,97	1,86	1,88	1,86	1,95	1,94

(4) La charge des CPUs est plus équilibrée entre les deux CPUs lorsque le temps de la communication est négligeable. La présence de délais de communication rend donc le problème de parallélisation plus difficile.

Donc, lorsque les durées de communication ne sont pas très grandes par rapport aux durées des tâches, notre heuristique est performante si on adopte la stratégie "MinMin" et la règle de priorité du chemin le plus long "Dijkstra". Pour savoir s'il y a des variantes selon le type de graphe nous avons effectué des analyses additionnelles sur les graphes avec des communications nulle et la stratégie d'affectation "MinMin".

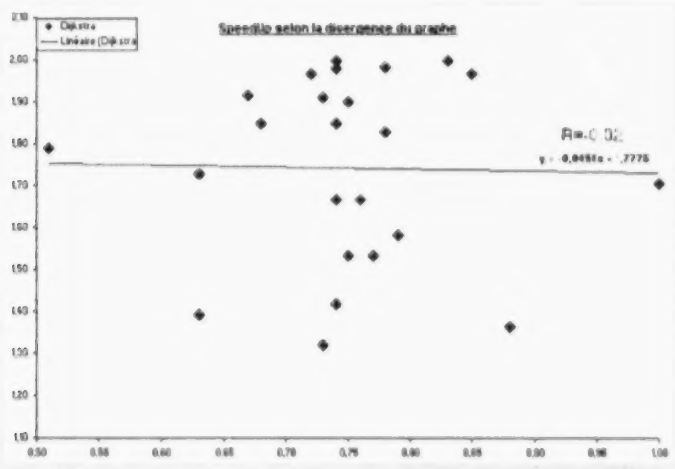

Figure 5.1 : Variation du "SpeedUp" moyen selon la divergence du graphe sur les graphes-tests avec la priorité Dijkstra (communication nulle).

La figure 5.1 montre la variation du "SpeedUp" moyen en fonction de la divergence pour la meilleure solution avec la stratégie "MinMin" et la priotité "Dijkstra". On a une droite de tendance quasiment horizontale mais avec un coefficient de corrélation pratiquement de zéro ($R = -0.02$). On peut donc conclure que le SpeedUp est indépendant de la divergence du graphe avec notre heuristique.

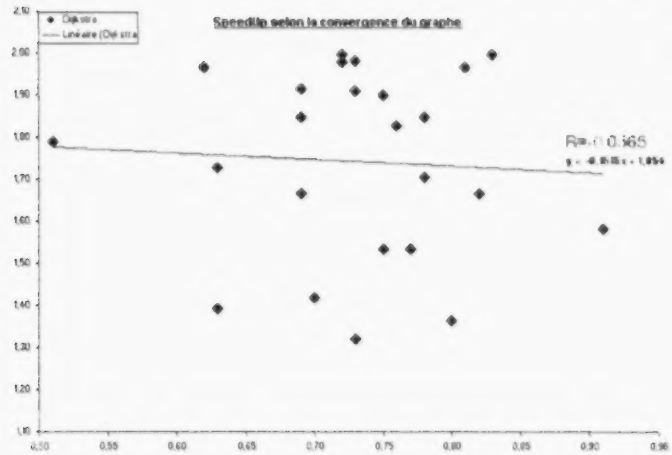

Figure 5.2 : Variation du "SpeedUp" moyen selon la convergence du graphe sur les graphes-tests avec la priorité Dijkstra (communication nulle).

La figure 5.2 montre le "SpeedUp" pour les mêmes solutions de la figure 5.1 en fonction de la convengence du graphe, conduit à la même conclusion : le SpeedUp est indépendant de la convergence du graphe.

Par contre, pour les figures 5.3 et 5.4 qui représentent la variation du "SpeedUp" en fonction du nombre de tâches et de la densité du graphe respectivement, on a des tendances assez prononcées. Le "SpeedUp" croît avec le nombre de tâches ($R = +0.5264$) mais décroît avec la densité du graphe ($R = -0.6787$).

Donc, trois points important sont à constater: (1) La solution ne dépend pas de la divergence ou convergence du graphe; (2) plus le graphe est dense, moins on peut profiter de la parallélisation; (3) la parallélisation semble plus facile quand le nombre de tâches augmente, lorsque la distribution des communications est relativement homogène.

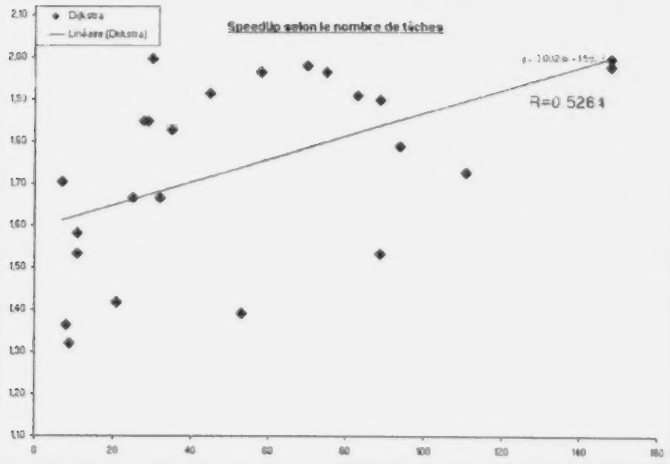

Figure 5.3 : Variation du "SpeedUp" moyen selon le nombre de tâches du graphe sur les graphes-tests avec la priorité Dijkstra (communication nulle).

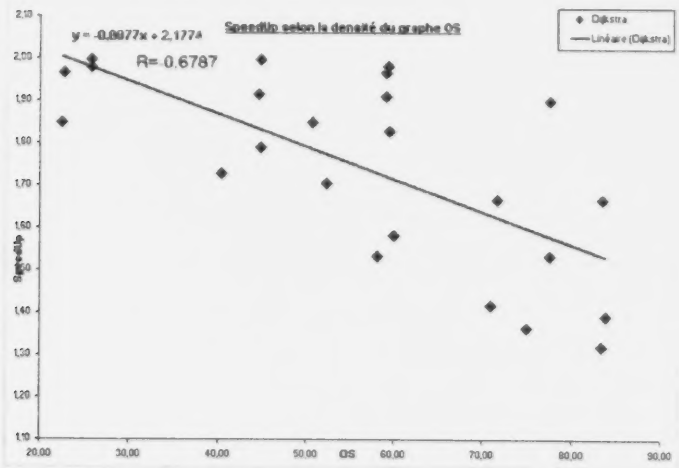

Figure 5.4 : Variation du "SpeedUp" moyen selon la densité du graphe sur les graphes-tests avec la priorité Dijkstra (communication nulle).

À la lumière des résultats, il faut implanter la règle "MinMin" et dijkstra. Pour des fins de robustesse on conserve les six règles de priorité dans notre heuristique. Toutefois, les temps de communication (types de communication Nulle versus Équivalente) ont un impact sur la performance de notre algorithme. Pour améliorer la robustesse de notre heuristique, nous avons décidé d'ajouter une simple version d'une heuristique de "Clustering" (Gerasoulis et Yang, 1993), en plus des six règles de priorité (voir section sur les tests pratiques), qui consiste à regrouper certains noeuds du DAG avant de les affecter aux différents CPUs. De plus, dans l'environnement Rt-lab, on peut générer des DAG de différentes granularité pour un même modèle Simulink afin de trouver le DAG générant la meilleure solution avec notre heuristique de priorité.

5.2 Tests pratiques

Après avoir évalué notre heuristique d'ordonnancement, nous testons tous nos modules depuis l'analyse du modèle Simulink séquentiel jusqu'à l'exécution du modèle parallèle dans l'environnement Rt-lab en passant par la génération du DAG. Le test porte sur un modèle de contrôle de système réel. L'objectif est de démontrer la faisabilité de notre logiciel d'automatisation, le respect de l'intégrité du modèle de départ et la performance de notre heuristique. Ce test a fait l'objet d'un article qui a été publié au Transactions de la Société Canadienne du Génie Mécanique. La référence complète de l'article est :

Rabbath, C.A., Lechevin, N., Ait El Cadi, A., Lapierre, S., Abdone, M. and Crainic, T., Automatic Parallelization of Electro-Mechanical Systems Governed by ODEs, Transactions of the CSME, Vol.26, No.3, 2002.

L'article est joint à l'annexe II. Nous présentons les points importants de ce test.

5.2.1 Objectif du test

Le premier but de ce test est de vérifier que le modèle de simulation n'est pas altéré, c.-à-d. que nous respectons bien l'intégrité du modèle. Il faut que notre modèle parallèle et le modèle original séquentiel soient interchangeable car sinon, si la réponse de la simulation de notre modèle est décalée par rapport à l'original, notre outil d'automatisation serait inutile et l'utilisateur ne pourra pas en profiter.

Le deuxième but du test est d'évaluer l'apport de notre outil en termes d'accélération du processus de parallélisation d'un modèle et d'amélioration de la qualité de la parallélisation. Finalement, on veut vérifier la faisabilité technologique de notre approche.

5.2.2 Objet du test

Les tests portent sur deux modèles de robots "Multi-body Systems", modèles développés par Opal-rt à la demande de l'Agence Canadienne de l'Espace. Le premier est un robot avec un bras à deux degrés de liberté. La figure 5.5 montre ce bras à double articulation (a) et l'actionneur (b). Le modèle Simulink associé est montré à la figure 5.6. Le deuxième robot est composé de deux bras manipulateurs avec trois degrés de liberté. Chacun des deux bras manipulateurs portent une boîte entre les deux, comme sur la figure 5.7, et doivent exécuter certains mouvements sans que la boîte tombe ou soit écrasée par manque de synchronisation. La figure 5.8 montre le niveau supérieur du modèle Simulink associé à ce robot.

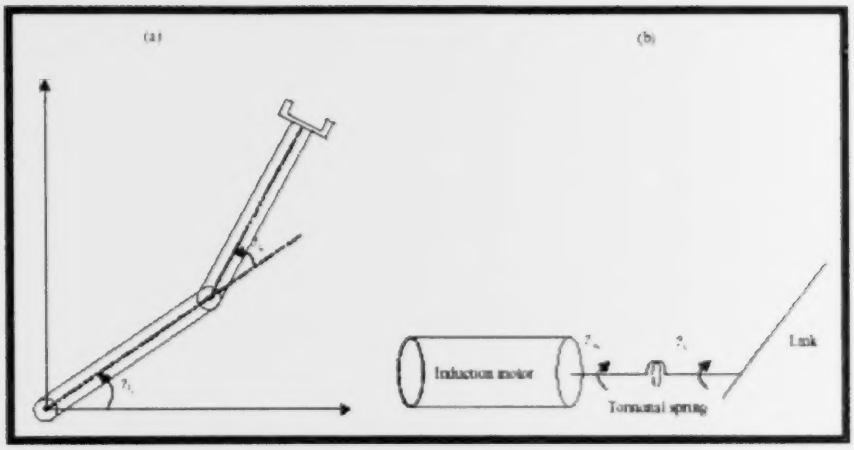

Figure 5.5 : (a) Bras à deux degrés de liberté; (b) l'actionneur

5.2.3 Résultats et conclusion

Concernant la faisabilité technologique, les tests on pu être menés jusqu'au bout et toute la boucle d'automatisation a pu être testée. En plus, pour le modèle de la figure 5.7, notre heuristique l'a séparé en quelques secondes, générant une aussi bonne solution qu'un ingénieur expérimenté ayant passé deux jours pour effecture la même séparation (le test a été fait chez Opal-rt). Notre logiciel est donc un succès sur le plan de la faisabilité.

L'heuristique utilisée dans cette section est celle avec les règles de priorité et avec la méthode de "Clustering". Nous présentons ici une analyse détaillée des deux solutions de parallélisation proposées par la règle de priorité du chemin le plus long "Dijkstra" et de la méthode de "Clustering".

Le tableau 5.7 montre les résultats pour cinq tests effectués sur le modèle à un robot (M1.X) et le modèle à deux robots (M2). On donne les temps C_{max} de la solution à 1 ou 2 CPUs et le facteur d'accélération. Les résultats de la parallélisation obtenus

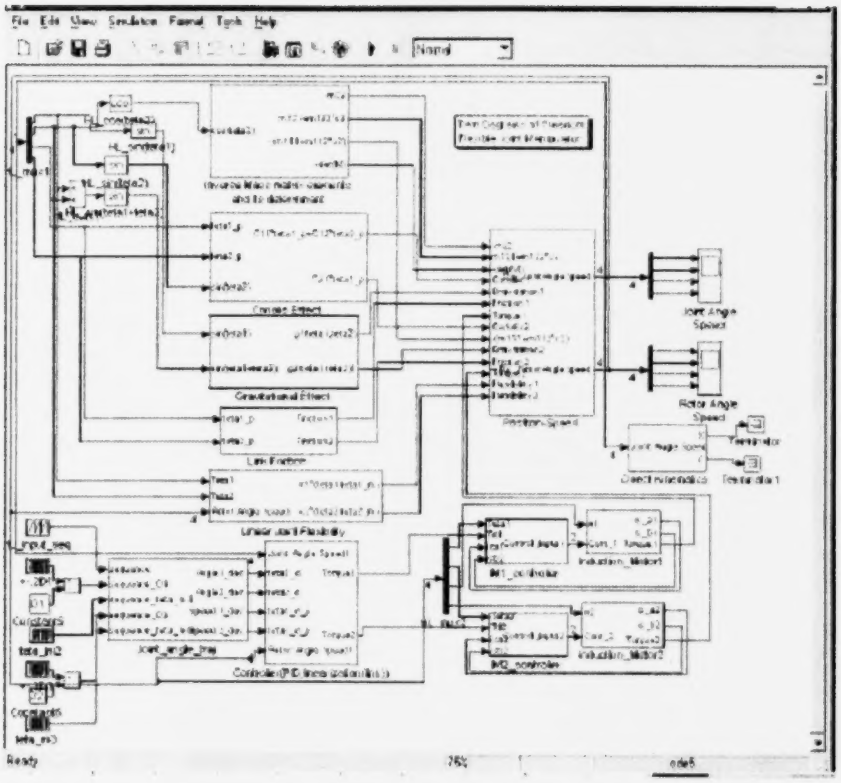

Figure 5.6 : Niveau supérieur du modèle Simulink du robot à deux degrés de liberté de la figure 5.5.

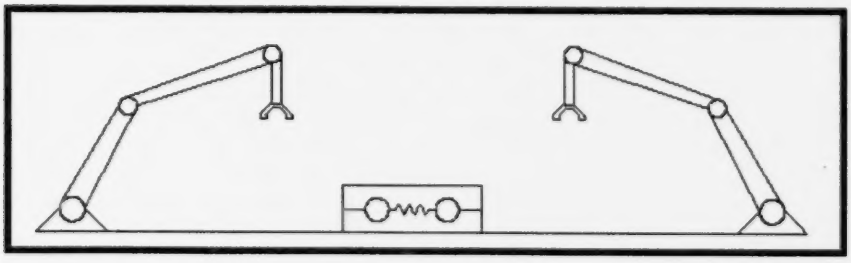

Figure 5.7 : Deux manipulateurs exécutant une tâche de contact.

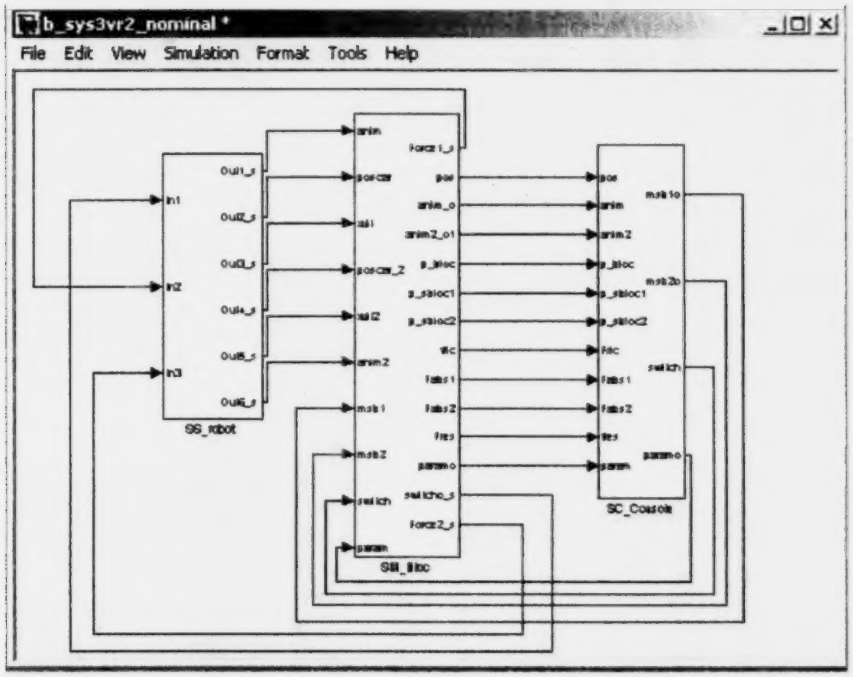

Figure 5.8 : Niveau supérieur du modèle Simulink du robot de la figure 5.7.

Tableau 5.7 : Temps d'exécution et facteur d'accélération par modèle et par solution (le M1. fait référence à des variantes du robot à deux degrés de liberté et le modèle M2 fait référence au système des deux manipulateurs. Le C_{max} est indiqué en gras.)

		1 CPU	2 CPU*		
Model	Solution	(ms)	CPU1 (ms)	CPU2 (ms)	Speed Up
M1.1	Dijkstra	0,939	0,700	0,246	1,34
	Clustering	0,939	0,694	0,235	1,35
M1.2	Dijkstra	1,051	0,589	0,320	1,78
	Clustering	1,051	0,375	0,688	1,52
M1.3	Dijkstra	1,051	0,472	0,580	1,80
	Clustering	1,051	0,458	0,589	1,78
M1.4	Dijkstra	1,225	0,733	0,291	1,40
	Clustering	1,225	0,813	0,399	1,50
M2	Dijkstra	0,100	0,054	0,057	1,75
	Clustering	0,094	0,057	0,054	1,75

* Les temps d'exécution sontmesurés par les ingénieurs d'Opal-rt. Il peut y avoir des erreurs sur les valeurs mesurèes à cause de la méthode utilisèe - on n'a pas tout le temps le contrôle totale du processeur.

Tableau 5.8 : Résultat pour une séparation sur 3 CPU pour le modèle M1.5 (Robot à deux degrés de liberté). Le C_{max} est indiqué en gras.

		1 CPU	3 CPU			
Model	Solution	(ms)	CPU1 (ms)	CPU2 (ms)	CPU3 (ms)	Speed Up
M1.5	Clustering	1,225	0,161	0,161	0,802	1,52

avec notre heuristique sont très bons. En fait, les facteurs d'accélération sont proches et parfois mieux que ceux obtenus par les ingénieurs d'Opal-rt. Le tableau 5.8 montre la performance d'une parallélisation sur trois CPUs. Encore une fois, la solution est aussi bonne que celle proposée par un ingénieur d'expérience.

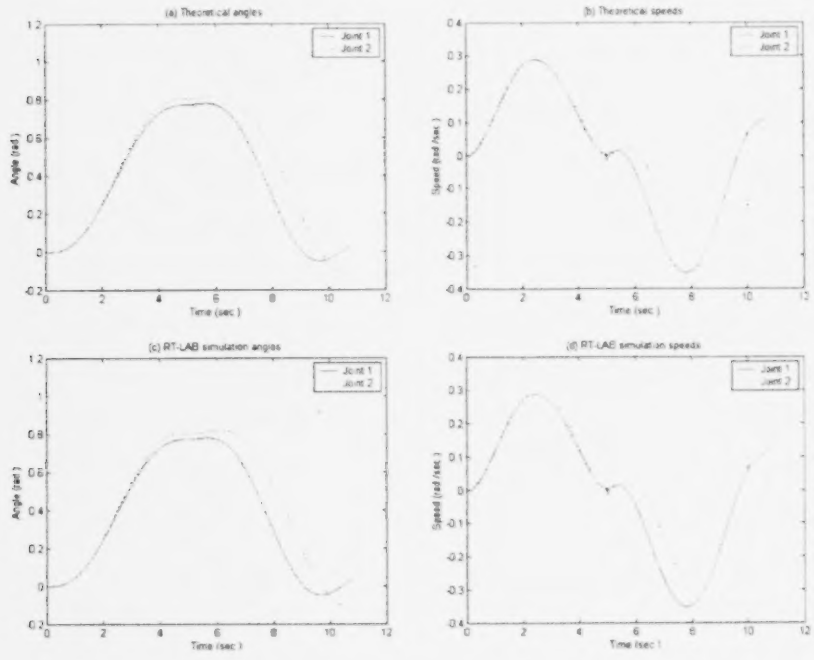

Figure 5.9 : Comparaison, pour le robot à deux degrés de liberté, entre les résultats prédits par l'heuristique et les résultats réels une fois le modèle séparé est exécuté sur Rt-lab.

Pour la granularité, les tests théorique disaient que la solution s'améliore quand le nombre de tâches augmente. Au tableau 5.7, on remarque que le facteur d'accélération du modèle M1.3 est meilleur que celui du modèle M1.1. Or la granularité du modèle M1.1 est plus fine que celle du modèle M1.3. Ceci est contraire aux résultats théoriques rencontrés dans la littérature. Notre heuristique n'est donc pas capable de profiter d'un DAG plus détaillé afin de proposer des solutions de parallélisation

plus performantes.

Pour la précision et le respect de l'intégrité du modèle de départ, la figure 5.9 nous montre les résultats de comparaison entre les réponses du modèle séquentiel du robot de deux degré de liberté (a) et (b) et ceux du même modèle après parallélisation. Il n'y a aucun décalage de réponse entre le modèle de départ et le modèle d'origine.

Il nous reste deux problèmes à régler : Rt-lab ne respecte pas la séquence fournie par l'heuristique et le temps des tâches est un estimé. En effet, sur Rt-lab, on peut spécifier seulement l'ensemble des tâches exécutées par chaque CPU, mais pas la séquence de leur exécution. Bien sûr, le séquenceur de Rt-lab respecte les contraintes de précédence mais deux tâches indépendantes sont exécutées dans un ordre quelconque. En conséquence, l'horaire (et le "SpeedUp" prédit par l'heuristique) est différent de la réalité. Le tableau 5.9 montre cet écart pour les tests qu'on a mené sur le premier robot. On y voit bien que le "SpeedUp" prédit est généralement plus grand que la réalité. Il faut donc vérifier la robustesse de notre solution au séquenceur, ce qui est fait à la section suivante de ce chapitre.

Tableau 5.9 : Écart entre les résultats prédits par l'heuristique et les valeurs obtenu réellement.

		SpeedUp	
Model	Solution	Prédit	Réel
M1.1	Dijkstra	1,37	1,34
	Clustering	1,34	1,35
M1.2	Dijkstra	1,33	1,78
	Clustering	1,98	1,52
M1.3	Dijkstra	1,82	1,80
	Clustering	1,96	1,78
M1.4	Dijkstra	1,40	1,40
	Clustering	1,98	1,50
M1.5	Clustering	2,08	1,52

Globalement on peut dire que les résultats pratiques sont concluants. L'automatisation est faisable, on a des résultats très performants dans des délais plus que satisfaisants par rapport à la procédure manuelle. Toutefois les tests ont montrés qu'il reste du travail à faire. L'heuristique est moins performante sur un DAG de plus petite granularité. L'ajout d'une métaheuristique devrait pouvoir nous permettre d'obtenir le résultat contraire : une meilleure solution pour le DAG détaillé. C'est que notre heuristique est myopique et prend plus de mauvaises décisions lorsqu'il y a trop d'informations et que les temps de communication sont hétérogènes. Autre point, des tests préliminaires montrent que, pour le modèle du robot à deux degré de liberté, on obtient un facteur d'accélération de **1.87** (valeur réelle) si on duplique certaines tâches, plutôt que **1.7**. Colin et Chrétienne (1991) avaient démontré la valeur de la duplication des tâches dans le contexte de parallélisation. Les gents d'Opal-rt étaient sceptiques sur ce résultat mais leurs propres tests leurs ont prouvés le contraire. Il faudra investiguer cette option plus en profondeur dans un futur projet de recherche.

5.3 Tests de robustesse

Les résultats jusque là sont très concluants. Mais il reste le problème que l'environnement de Rt-lab n'est pas totalement sous contrôle. Le séquenceur de Rt-lab n'accepte pas qu'on lui impose une séquence d'exécution des tâches. Donc le but des tests de cette section est de vérifier la robustesse de nos solutions vis-à-vis de cet environnement, de justifier certaines limitations et puis d'évaluer la valeur de l'approche métaheuristique.

5.3.1 Objectif du test

Le but de ce test est de démontrer la robustesse de nos solutions vis-à-vis d'un séquenceur non contrôlable. Autrement dit, il faut vérifier que l'écart entre la valeur prédite pour le facteur d'accélération et la valeur réelle reste acceptable.

5.3.2 Les données pour le test

Les données des tests de robustesse sont les graphes du test de la première section de ce chapitre. Les mêmes graphes sont étudiés avec les mêmes scénarios car on essaie d'évaluer la robustesse de chacune des stratégies d'affectation et de chacune des règles de priorité. Pour ce, on a eu besoin de développer un simulateur (en script m) qui joue le rôle d'un séquenceur non contrôlable (pour simuler Rt-lab). Ce simulateur prend en entrée la solution de chaque cas puis essaie de reconstruire l'horaire de manière complètement aléatoire, sans respect de la séquence de départ, le tout, bien sûr, en tenant compte de toutes les contraintes du graphe. Notre simulateur est plus imprévisible que Rt-lab, rendant nos essais encore plus valides que si on les avait effectuées seulement sur Rt-lab.

5.3.3 Déroulement du test

Les tests consiste à prendre chacune des 1296 affectations (24 graphes, 3 types de communication, 3 stratégies d'affectation et 6 règles de priorité) et ainsi générer différents horaires par le simulateur. Les données suivantes sont collectées ou calculées avec ces 100 horaires :

(1)Pour le MakeSpan : la moyenne, le min, le max, la médiane et l'écart-type.

(2)Pour le SpeedUp : on calcule l'écart relatif moyen par rapport à la valeur

prédite par notre heuristique. On détermine aussi l'intervalle de confiance sur cette moyenne à 95 %.

5.3.4 Les résultats du test

Les principales valeurs à retenir de nos tests sont la moyenne des écarts relatifs sur le "SpeedUp" et l'intervalle de confiance à 95% sur cette dernière. Ils sont respectivement montrés dans les tables 5.10 et 5.11.

Tableau 5.10 : Moyenne des écarts relatifs entre le facteur d'accélération prédit et les valeurs obtenus par le simulateur.

Type de		Moyenne des écarts relatifs entre le facteur d'accélération					
Communication	Stratégie	Dijkstra	GraphLevel	TaskLength	Random	SumChild	NbChild
Equivalente	Alea	9,4%	6,6%	4,6%	2,8%	5,0%	5,4%
	MinMax	9,5%	4,3%	5,9%	5,2%	5,3%	4,4%
	MinMin	11,6%	5,6%	4,3%	4,7%	6,0%	7,6%
Nulle	Alea	8,8%	5,8%	4,4%	4,3%	5,3%	6,1%
	MinMax	3,6%	6,8%	4,8%	5,3%	3,8%	3,5%
	MinMin	12,3%	5,4%	5,4%	4,2%	6,5%	8,6%
prépondérente	Alea	0,3%	0,2%	0,2%	0,2%	0,1%	0,2%
	MinMax	0,2%	0,1%	0,2%	0,1%	0,1%	0,1%
	MinMin	0,7%	0,7%	0,4%	0,3%	0,4%	0,4%
Moyenne		6,3%	3,9%	3,3%	3,0%	3,6%	4,0%

Premièrement, si on regarde la table 5.11, on remarque que l'intervalle de nos résultats est très faible - au plus 1%. Il y a donc très peu de variablité au niveau des horaires : il y a des horaires "dominants".

Si on revient à la table 5.10, on constate que l'écart est relativement petit - 12.3% au maximum - mais dépend du type de solutions utilisées. Nous remarquons, en général, que plus le problème est contraint plus l'écart est petit. Pour les cas de communication prépondérente, la variation est de moins de 1%. De plus, plus la solution est bonne plus l'écart est grand. D'ailleurs, dans la section 1, on

a eu les meilleurs résultats pour "Dijkstra" avec la stratégie "MinMin" et c'est pour ce même cas qu'on a les plus grands écarts. Donc, on en déduit que Rt-lab limite la performance de notre affectation en faisant des choix non optimaux d'ordonnancement.

Puisque Rt-lab change notre ordonnancement, nous voulons savoir si notre meilleur solution prédite doit toujours être celle choisie. Pour ce faire, il faut comparer les temps d'exécution total des horaires générés par notre simulateur afin de voir si notre meilleure solution reste dominante dans le contexte d'un moins bon séquenceur. Nos tests démontrent que la meilleure solution prédite reste la meilleure dans 88.0% des cas (190 fois sur 216 graphes). Donc, nous n'avons pas besoin d'ajouter notre simulateur à notre heuristique : on choisit l'affectation générant le meilleur horaire prédit.

Tableau 5.11 : Intervalle de confiance à 95% sur les moyennes de la table 5.10.

Type de		Rayon de l'intervalle de Confiance à 95%					
Communication	Stratégie	Dijkstra	GraphLevel	TaskLength	Random	SumChild	NbChild
Equivalente	Alea	0,9%	1,1%	1,1%	0,9%	1,0%	1,0%
	MinMax	0,9%	1,0%	1,0%	1,0%	1,0%	0,9%
	MinMin	1,0%	0,8%	1,0%	0,9%	0,9%	1,2%
Nulle	Alea	0,9%	1,0%	1,0%	1,0%	1,0%	1,0%
	MinMax	0,6%	1,0%	1,0%	1,0%	0,8%	0,6%
	MinMin	1,0%	0,9%	0,9%	0,9%	0,9%	1,0%
Prependerente	Alea	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%
	MinMax	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%
	MinMin	0,1%	0,1%	0,1%	0,1%	0,1%	0,1%
Moyenne		0,6%	0,7%	0,7%	0,6%	0,7%	0,6%

5.3.5 Conclusion

D'après ce qui précède, on peut conclure que notre heuristique est relativement robuste à l'ordonnanceur (un écart maximum de 12.3%). Même si la meilleure

solution a le plus de chance d'être moins bien ordonnancéepar Rt-lab, elle reste la solution dominante. On conserve donc notre choix d'affectation optimale, pas besoin d'utiliser notre simulateur pour avoir la meilleure d'affectation.

CONCLUSION

L'objectif de ce travail est de démontrer la faisabilité du projet de l'automatisation de la parallélisation d'un modèle Simulink dans l'environnement Rt-lab. Nous pouvons confirmer que l'objectif est atteint. La méthode utilisée consiste en deux phases. La première est basée sur la traduction du modèle Simulink en un graphe orienté acyclique - le DAG - qui représente d'une manière formelle l'ensemble des tâches à exécuter avec leurs contraintes de communication et de précédence. La deuxième phase s'occupe de la séparation de ces dites tâches sur un nombre de CPUs donné.

En effet, dans un premier temps, nous avons cherché à formaliser notre problème. Et bien sûr la meilleure façon était de le représenter le modèle Simulink sous forme de DAG. On avait le choix entre deux approches : celle basée sur le code C représentant le modèle et celle basée sur le schéma fonctionnel de Simulink. Notre choix s'est arrêté sur l'approche basée sur le schéma fonctionnel pour deux raisons. Premièrement, tout le projet est réalisé dans l'environnement de Matlab et le shell de ce dernier offre tous les moyens nécessaires pour créer un parseur idéal des modèles Simulink. Deuxièmement, le schéma fonctionnel contient beaucoup d'informations qu'on risque de perdre une fois le modèle traduit en code C. Car, ce qui est le cas des générateurs de codes, le code généré est généralement très redondant et contient quelque fois des instructions parasites.

Pour le problème de parallélisation, nous avons choisi une méthode de résolution simple ayant fait ses preuves et facile à implanter. Notre choix s'est arrêté sur une heuristique basée sur les règles de priorité en plus de quelques techniques de "Clustering". En essayant différentes stratégies d'affectation et différentes règles de priorité, nous avons trouvé que la méthode la plus efficace pour notre cas est celle

basée sur la règle du chemin le plus long (en utilisant l'algorithme de "Dijkstra") et qui affecte les tâches au processeur le moins chargé.

Les résultats sont très concluants. Une première phase de tests a pu démontrer l'efficacité de notre heuristique. Une deuxième phase de tests pratiques exécutées sur deux robots a pu démontrer la faisabilité du projet ainsi que la performance de nos solutions. En effet, notre procédure automatique a eu des résultats aussi performants qu'un ingénieur expérimenté qui procède manuellement mais dans des proportions de temps incomparables (quelques secondes contre deux jours). De même, les facteurs d'accélération dans des cas de séparation sur deux CPU ont approché les 1.8. La troisième phase a servi à montrer la robustesse de notre solution vis-à-vis d'un ordonnanceur non contrôlable tel que Rt-lab. La meilleure solution reste la meilleure lorsque l'ordonnanceur ne fait pas les choix optimaux.

Toutefois, même si l'approche est très intéressante, les performances de l'heuristique de séparation restent très modestes dans un cas général. Notre heuristique est myope et prend plus de mauvaises décisions lorsqu'il y a trop d'informations. Des tests ont montré que lorsque le graphe est très dense ou la communication très grande les résultats sont moins bons. L'heuristique de "Clustering" donne des résultats intéressants mais il faudrait travailler plus en profondeur ce type d'algorithme.

RECOMMANDATIONS ET TRAVAUX FUTURS

À la suite de notre conclusion nous recommandons, quant au problème de séparation, une exploration du monde des métaheuristiques tel que le tabou et le génétique. Ces méthodes n'ont pas le problème de myopie comme le cas de notre heuristique, ce qui les rend beaucoup plus robustes dans les cas de graphe de fine granularité et

de forte densité. De même, nous recommandons l'exploration du "Clustering" et de la "Duplication". Les tests ont prouvé l'utilité de ces téchniques. Parallèlement, il faudrait analyser plus en profondeur le potentiel des heuristiques de "Clustering" et peut être même utiliser cette heuristique à l'intérieur d'une métaheuristique.

Quant à la duplication des tâches nous recommandons à Opal-rt d'explorer la possibilité d'offrir cette option dans Rt-lab. Les tests préliminaires concordent avec les publications scientifiques quant à ses bénéfices. Ainsi, il faudrait modifier les heuristiques et les métaheuristiques afin de trouver des horaires permettantla duplication des tâches.

Du côté de la génération du graphe, il reste encore des travaux à faire pour tenir compte de tous les cas particuliers du modèle Simulink. Principalement il faut voir comment traiter :

(1) Les blocs "S-functions" : en combinant avec des techniques de parallélisation basée sur le code;

(2) Les blocs d'habillage tels que le "Mux" ne représentant aucune quantité de calcul mais qui, si mal positionnés par le concepteur, risquent de cacher des opportunités de parallélisme : pour ces derniers la solution recommandée est de détecter ces blocs et de les défaire.

(3) Les "Multi-rate" sous-système : c'est le cas où notre modèle est composé de plusieurs sous-systèmes dont le pas d'intégration est différent. Dans ce cas, on sait que les pas sont multiples les uns des autres. Il faut donc se baser sur le plus grand pas puis dupliquer les sous-systèmes des autres.

(4) Les "enabled/triggered" sous-systèmes : ce sont des sous-systèmes qui ne sont pas exécutés durant tous les pas de simulation mais seulement sous certaines conditions. Dans ce cas, il faut voir s'il y a un lien d'exclusion entre ces sous-systèmes sinon il faut travailler avec le pire des cas, c'est-à-dire le pas de simulation le plus chargé.

Il y a aussi des contraintes physiques qu'il faudra intégrer dans la génération du DAG. C'est le cas où un des modules de la simulation devrait servir ultérieurement comme une composante dans un ensemble réel.

La logique de base de l'automatisation de la parallélisation d'un modèle Simulink est développée et est viable technologiquement. Il reste toutefois un peu de recherche et de développement avant de commercialiser notre logiciel.

RÉFÉRENCES

ARCUS, A. L. (1966). Cosmoal: A computer method of sequencing operations for assembly lines. *International Journal od Production Research*, **4**, 259–277.

BEAUGENDRE, P. (1995). Parallélisation du code mimosa. Rapport de stage, Institut de Recherche en Informatique et Systèmes Aléatoire (I.R.I.S.A.).

BELLEN, A. et ZENNARO, M. (1989). Paralle algorithms for initial value problems for difference and differential equations. *Journal of Computational and Applied Mathematics*, **25**, 341–350.

BLAZEWICZ, J., CELLARY, W., SLOWINSKI, R. et WĘGLARZ, J. (1986). *Scheduling Under Resource Constraints - Deterministic Models*. J.C. Baltzer AG, Basel.

BLAZEWICZ, J., DROZDOWSKI, M. et ECKER, K. (2000). Management of resources in parallel systems. In BLAZEWICZ, J., ECKER, K., PLATEAU, B. et TRYSTRAM,D., editors, *Handbook on Parallel and Distributed Processing*, pages 263–341. Springer-Verlag, Berlin.

BLAZEWICZ, J., ECKER, K., SCHMIDT, G. et WĘGLARZ, J. (1993). *Scheduling in Computer and Manufacturing Systems*. Spriner-Verlag, Berlin, Germany.

BLAZEWICZ, J., LENSTRA, J. K. et RINNOOY KAN, A. H. G. (1983). Scheduling subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, **5**, 11–24.

BOCTOR, F. F. (1995). A multiple-rule heuristic for assembly line balancing. *Journal of the Operational Research Society*, **49**, 3–13.

BRUCKER, P. (1995). *Scheduling Algorithms*. Springer-Verlag, Berlin.

BURRAGE, K. (1995). *Parallel and sequential methods for ordinary differential equations*. Clarendon press. Oxford.

BUTCHER, J. C. (1993a). Diagonally-implicit multi-stage integration methods. *Applied Numerical Mathematics*, **11**, 347–363.

BUTCHER, J. C. (1993b). General linear methods for the parallel solution of ordinary differential equations. *Wold Scientific series in Applicable Analysis*, **2**, 99–111.

CARLIER, J. et PINSON, E. (1989). An algorithme for solving the job-shop problem. *Management Science*, **35**, 164–176.

CERNY, V. (1985). Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications*, **45**, 41–51.

CHANDY, K. M. et MISRA, J. (1979). Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on software engineering*, **SE-5**(5), 440–452.

CHARTIER, P. (1994). L-stable parallel one-block methods for ordinary differential equations. *SIAM Journal on Numerical Analysis*, **31**(2), 552–571.

CHARTIER, P. et PHILIPPE, B. (1993). A parallel shooting technique for solving dissipative ode's. *Computing*, **51**(3-4), 209–236.

CHEN, W. H. et LIN, C. S. (2000). A hybrid heuristic to solve a task allocation problem. *Computers and Operations Research*, **27**(3), 287–303.

CHIANG, W. C. (1998). The application of a tabu search metaheuristic to the assembly line balancing problem. *Annals of Operations Research*, **77**, 209–227.

CHRÉTIENNE, P. (1989). A polynomial algorithm to optimally schedule tasks on a virtual distributed system under tree-like precedence constraints. *European Journal of Operational Research*, **43**, 225–230.

COGNÉ, L. (1997). *Simulation de systèemes physiques lagrangiens : de la représentation symbolique aux évaluation numérique séquentielles et parallèles.* Thèse de doctorat, Université de Rennes 1, Rennes, France.

COLIN, J. Y. et CHRÉTIENNE, P. (1991). C.p.m. scheduling with small communication delays and task duplication. *Operations Research*, **39**(4), 680–684.

DADO, B., MEÒHART, P. et SFAÍK, J. (1993). Distributed simulation: A simulation system of discrete event systems. in: Cosnard,m., puigjaner,r. (eds.): Decentralized and distributed systems. *IFIP Transactions.*, **A-39**, 343–353.

DARTE, A., ROBERT, Y. et VIVIEN, F. (2000). *Scheduling and Automatic Parallelization.* Birkhauser, Boston.

EL-REWINI, H., LEWIS, T. G. et ALI, H. (1994). *Task Scheduling in parallel and distributed systems.* Prentice-Hall, New York.

FRENCH, S. (1982). *Sequencing and scheduling: an introduction to the mathematics of the job-shop.* John Wiley, New York.

GAREY, M. R. et Johnson,D. S. (1979). *Computers and Interactability: A guide to the theory of NP-Completeness.* Freeman, San Francisco, CA.

GEAR, C. (1986). Paralle methods for ordinary differential equations. Technical report R-87-1369, University of Illinois.

GERASOULIS, A. et YANG, T. (1993). On the granularity and clustering of directed acyclic task graphs. *IEEE Transactions on Parallel and Distributed Systems*, **4**(6), 686–701.

GHOSH, S. et GAGNON, R. J. (1989). A comprehensive literature review and analysis of the design, balancing and scheduling of assembly lines. *International Journal of Production Research*, **27**, 637–670.

GLOVER, F. et LAGUNA, M. (1997). *Tabu search*. Kluwer Academic Publishers, Boston.

KARAYANAKIS, N. M. (1995). *Advanced System Modelling and Simulation with Block Diagram Languages*. CRC Press.

KIRKPATRICK, S., GELATT, C. D. et VECCHI, M. P. (1983). Optimization by simulated annealing. *Science*, **220**, 671–680.

LAPIERRE, S. D. et RUIZ, A. B. (2001). Balancing assembly lines with multi-attributed tasks. Working paper $\sharp C7PQMRPO2001 - 19 - X$, CRT, Montreal.

LAPORTE, G., GENDREAU, M., POTVIN, J. Y. et SEMET, F. (2000). Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research*, **7**, 285–300.

LAWLER, E. L., LENSTRA, J. K. et RINNOOY KAN, A. H. G. (1982). Recent developments in deterministic sequencing and scheduling: A survey. In Dempster, M.A.H., LENSTRA, J.K. et RINNOOYKAN,A.H.G., editors, *Deterministic and Stochastic Scheduling*. Reidel, Dordrecht, The Neaderlands.

LAWLER, E. L., LENSTRA, J. K., RINNOOY KAN, A. H. G. et SHMOYS, D. B. (1993). Sequencing and scheduling: Algorithms and complexity. In Graves, S.C., Rinnoy Kan, A.H.G. et Zipkin, P.H., editors, *Logistics of production and inventory*, volume 4 of *Handbooks in Operations Research and Management Science*, pages 445–522. North-Holland, Amsterdam.

LEE, C. Y., PIRAMUTHU, S. et TSAI, Y. K. (1997). Job shop scheduling with a genetic algorithm and machine learning. *International Journal of Production Research*, **35**, 1171–1191.

MACULAN, N., PORTO, S. C. S., RIBEIRO, C. C. et DE SOUZA, C. C. (1999). A new formulation for scheduling unrelated processors under precedence constraints. *RAIRO*, **33**, 87–92.

MARIC, S. et JOVANOVIC, Z. (1999). Dynamic task scheduling with precedence constraints and communication delays. In *Parallel Computing Technologies. 5th International Conference*, pages 77–91. PaCT-99.

MATHWORKS (2000a). *Matlab User's Guide.*

MATHWORKS (2000b). *Real Time Workshop User's Guide.*

MATHWORKS (2000c). *Simulink User's Guide.*

MILON, O. (1999). Gestion de projet avec contraintes de ressources. Master's thesis, École Polytechnique de MontrÉal.

MORI, M. et TSENG, C. C. (1997). A genetic algorithm for multi-mode resource constrained project scheduling problem. *European Journal of Operational Research*, **100**, 134–141.

MUNIER, A. et HANEN, C. (1997). Using duplication for scheduling unitary tasks on m processors with unit communication delays. *Theoretical Computer Science*, **178**, 119–127.

MURTY, K. G. (1994a). *Operations Research : Deterministic Optimization Models.* Prentice-Hall, Englewood Cliffs, NJ.

MURTY, K. G. (1994b). *Operations research: deterministic optimization models.* Prentice-Hall, Englewood Cliffs, NJ.

MUTH, J. F. et LUTHER, T. G. (1963). *Industrial Scheduling*. Prentice-Hall, Englewood Cliffs, NJ.

NOWICKI, E. et SMUTNICKI, C. (1996). A fast taboo search algorithm for the job shop problem. *Management Science*, **42**, 797–813.

OPAL-RT (2001). *RT-Lab User's Guide*.

RAULT, S. (1998). *Algorithmes parallèle pour le calcul d'orbites*. Thèse de doctorat, Université de Rennes 1, Rennes, France.

RUBINOVITZ, J. et LEVITIN,G. (1995). Genetic algorithm for assembly line balancing. *International Journal of Production Economics*, **41**, 343–354.

SAAD, R. (1995). Scheduling with communication delays. *Journal of Combinatorial Mathematics and Combinatorial Computing*, **18**, 214–224.

SARKAR, V. (1989). *Partitionning and scheduling parallel programs for execution on multiprocessors*. Cambridge, M.A.: M.I.T. Press.

SCHOLL, A. (1999). *Balancing and sequencing of assembly lines. 2nd ed.* Physica, Heidelberg.

SOLAR, M. et INOSTROZA, M. (2002). The scheduling problem in parallel programming. Montreal, CANADA. Optimization Days 02.

SPRECHER, A. (2000). Scheduling resource-constrained projects competitively at modest memory requirement. *Management Science*, **46**(5).

STONE, H. (1977). Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Transactions on Software Engineering*, **3**, 85–93.

STONE, H. (1987). *High-Performance Computer Architectures*. Reading, Mass. : Addison-Wesley.

TAILLARD, E. D. (1994). Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, **6**, 108–117.

THERSEN, A. (1998). Design and evaluation of tabu search algorithms for multiprocessor scheduling. *Journal of Heuristics*, **4**, 141–160.

URBAN, T. L. (1998). Optimal balancing of u-shaped assembly lines. *Management Science*, **44**(5).

VAN LAARHOVEN, P. J. M., AARTS, E. H. L. et LENSTRA, J. K. (1992). Job shop scheduling by simulated annealing. *Operations Research*, **40**, 113–125.

VIGO, D. et MANIEZZO, V. (1997). A genetic/tabu thresholding hybrid algorithm for the process allocation problem. *Journal of Heuristics*, **3**, 91–110.

ANNEXE I

LES DONNÉES ET LES RÉSULTATS DES TESTS

Tous les résultats et les données des tests sont disponibles sur la page web du projet à l'adresse :

http://www.crt.umontreal.ca/~aitelcad/Projets/Parallelisation/

ANNEXE II

ARTICLE : "PARALLÉLISATION AUTOMATIQUE DE SYSTÈMES ÉLECTRO-MÉCANIQUES BASÉS SUR DES ÉQUATIONS DIFFÉRENTIELLES ORDINAIRES"

Automatic Parallelization of Electro-Mechanical Systems Governed by ODEs [1]

C.A. Rabbath[1,2], N. Lechevin[3], A. Ait El Cadi[4], S. Lapierre[4], M. Abdoune[5] and T. Crainic[4]

[1]Defence Research and Development Canada - Valcartier 2459 Pie-XI N., Val-Bélair, PQ, Canada G3J 1X5, *Camille-Alain.Rabbath@drdc-rddc.gc.ca*

[2]Department of Mechanical Engineering, McGill University, Montréal, PQ, Canada H3A 2K6

[3]Hydro-Québec/NSERC Industrial Chair, Université du Québec à Trois-Rivières Trois-Rivières, PQ, Canada G9A 5H7

[4]Centre de Recherche sur les Transports, Université de Montréal, Montréal, PQ, Canada

[5]CAE Inc., Montréal, PQ,Canada, H4L 4X4

[1]Cet article a été publié au "Transactions of the Canadian Society of Mechanical Engineering"

Abstract The paper proposes an effective approach for the automatic parallelization of models of electro-mechanical systems governed by ordinary differential equations. The novel method takes a nominal mathematical model, expressed in block diagram language, and portions in parallel the code to be executed on a set of standard microprocessors. The integrity of the simulations is preserved, the computing resources available are efficiently used, and the simulations are compliant with real-time constraints; that is, the time integration of the ordinary differential equations is performed within restricted time limits at each iteration step. The proposed method is applied to a two-degree-of-freedom revolute joint robotic system that includes an induction motor and two inner-outer loop control laws. Numerical simulations validate the proposed approach.

Keywords: Electro-Mechanical Systems, Distributed Numerical Simulations, Parallel Processing, Heuristic, Directed Acyclic Graph.

Résumé Cet article propose une technique efficace pour la parallélisation de modèles de systèmes électro-mécaniques basés sur des équations différentielles ordinaires. La nouvelle méthode prend un modèle mathématique nominal, exprimé en schéma bloc, et partitionne le code en parallèle pour qu'il soit exécuté sur un ensemble de microprocesseurs standards. La cohérence des résultats de simulation est garantie, le matériel disponible est efficacement utilisé et les simulations rencontrent les contraintes de temps réel. La technique proposée est appliquée à un système robotique à deux degrés de liberté qui inclut un moteur à induction et deux boucles de commande. Des simulations numériques valident la méthode proposée.

Mots-clés: Systèmes électro-mécaniques, Simulations numériques distribuées, Traitement en parallèle, Heuristique, Graphe acyclique dirigé.

1.INTRODUCTION

Rapid development in computing technology over the last few decades has had an important impact on systems engineering. Complex systems, such as aeronautic and automotive systems, can now be simulated on desktop computers. Nevertheless, the computing demands of modern engineering analysis have also grown, to the extent that a single-processor computer, even with gigahertz clock speeds, is inadequate to execute real-time simulations of certain system designs. Real-time simulation refers to time integration of ordinary differential equations performed within restricted time limits at each iteration step. Cooperating robotic systems and generalized, electro-mechanical systems governed by ordinary differential equations fall into this class. The only way to achieve a high level of performance needed for these simulations is to distribute computations over a cluster of microprocessors [1].

The standard tools for systems design engineering are Mathworks' MATLAB and Simulink [2]. Opal-RT's RT-LAB [3] software and engineering simulators provide effective environments for real-time and distributed simulations of Simulink models on PCs. Importantly, RT-LAB software generally preserves simulation integrity while providing high-speed I/O and deterministic execution. These tools are particularly well suited to the context of rapid control prototyping. For example, as part of a complete design cycle, an engine control system can be entirely simulated in real-time. In order to do so, the model is obtained as a block diagram within Simulink, the code is generated with The Real-Time Workshop, and compilation of the code and the execution of the programs (real-time simulation of the model) is done with RT-LAB. Performances of a system can thus be verified and improved with the help of numerical simulations [4]. As another stage in the design cycle, with the automobile simulated on a set of standard PCs, the

performances achieved with the prototype electronic control unit, embedded on a chip, can be monitored. Alternatively, the control unit could be simulated on a standard PC and be connected to the *actual* car engine. For such systems, complex configurations can be implemented; for instance, drive-by-wire control systems can first be modeled and simulated over a PC network and then, as actual components are added, the simulated components can be replaced on a block-by-block basis. As a second example, in the field of gas-turbine engine control, simulations allow to fine tune the characteristics of the closed-loop turbine engine system based on pre-established performance criteria [5], [6]. Simulations allow the virtual integration of various modeled components of highly complex systems such as the inner- and outer-loop control laws, the fault tolerant management, the signal processing, and the fuel metering unit.

The main difficulty faced by engineers manipulating complex systems relates to the separation of mathematical models and algorithms. The engineer should be an expert in his/her field to choose a relatively good distribution of the model among the combinatorially-large number of possibilities while preserving model integrity. Besides the obvious process in which the topology of the electro-mechanical system dictates the separation [7], or equivalently the distribution of the given model, there is yet to be an efficient human-free, algorithmic-based process for parallelizing mathematical models of dynamic systems and for automatically distributing tasks on target machines.

The paper proposes an effective approach to automatically separate systems governed by ordinary differential equations with only limited human intervention. Two of the main characteristics of the technique are that it is well suited for mathematical models expressed as block diagrams and that it does not require user's knowledge of the model at hand. The novel scheme enables a parallel execution of the simulations on standard microprocessors using high-speed communications that

preserves the integrity of the simulation, efficiently uses the computing resources available, guarantees compliance with real-time constraints, and dramatically reduces the time to separate a model as compared to current manual methods. One way to measure the effectiveness of the proposed approach is to evaluate the fixed simulation (or iteration) step size, $T \in \Re^+$, achieved in the execution of the distributed simulation. On the one hand, it is important that T be large enough to allow for the required processor computing time, the inter-processor communications time and the overhead (process switching, synchronization, etc.). On the other hand, T must be short enough such that the distributed numerical simulations offer relatively high accuracy with respect to the theoretical behavior of the model at hand. For instance, it is well known that numerical integration with Euler's method [8] must be performed at the shortest T possible in order to obtain a relatively accurate approximation of the integration. The use of a relatively short T is particularly important in the simulation of systems exhibiting fast dynamics, such as induction motors, and discontinuous behavior, such as combustion engines.

The paper is divided as follows. Section 2 defines key concepts present in the fields of electro-mechanical system simulations and task allocation. Section 3 describes the proposed approach. Section 4 presents the robotic system under study. Numerical simulation results and a discussion are given in Section 5. Finally, concluding remarks are stated in Section 6.

2. PRELIMINARIES

Electro-mechanical systems can be described within the framework of block diagrams. This is particularly useful for control law design and numerical simulations. With the knowledge of the differential equations governing a system, an engineer can obtain a model and then simulate it on one or more target microprocessors. In this context, a model is a (simplified) graphical functional description of a given

dynamic system based on the observed behavior of the system at hand and on the established laws of physics. Models comprise blocks and subsystems. A block is a basic mathematical operation such as integration or multiplication. A subsystem encloses blocks that are grouped within a single graphical entity, usually a square block identified as a subsystem. Once the model of a dynamic system is known, it can be simulated. Simulation is defined as the solution of the equations describing a model, with respect to time, by computing the system's states and outputs. The fixed simulation step size (T) is the time period at which dynamic system computations are iterated. The Master (SM) and Slave (SS) subsystems of a model correspond to the physical target microprocessors upon which the simulations are distributed. Usually, the SM machine includes the hardware clock which synchronizes the execution on the targets. On each target machine, the effective step time (T_E) is the time to execute the computations associated with that portion of the model dedicated on the machine at hand and the inter-processor communications, for each time step. The total step time (T_T) corresponds to T_E in addition to the synchronization overhead, if any. The computation step time (T_C) is the time dedicated to perform all of the computations associated with a given model within a microprocessor, for each time step.

A useful alternative to the block diagram formalism is a Directed Acyclic Graph (DAG) [9]. A DAG is a non-unique representation of a model whose components are directed vertices (lines) and edges (circles) that are exempt of any cycle or circuit. The directed vertices act as precedence relationships between adjacent edges, or tasks, which represent a sequential execution in forward time. The tasks in a DAG may correspond to blocks (e.g. integration, gain multiplication of a signal, etc.) and subsystems of a model. In the DAG, a computational task, or simply a task, is an abstract object that corresponds to a non-empty set of blocks in a model, with finite duration in time (start and end times), and that has the following

attributes: name, identification number, type and time duration (computational weight). The granularity of a DAG is the smallest weight among the computational tasks with respect to the execution of the entire model. In the present work, the conventional DAG has been augmented with the information on the nature of data exchanges between tasks. A negative value in a directed vertex between two tasks means that the transfer is non-feedthrough, or equivalently is taking place between successive simulation steps, whereas a positive value implies a feedthrough transfer, or equivalently takes place within the step. In more details, a feedthrough task, such as a gain or a linear dynamic system expressed in state-space form as

$$\begin{aligned} &\tfrac{d}{dt}x(t) = Ax(t) + Bu(t) \\ &y(t) = Cx(t) + Du(t) \\ &u(t) : input \\ &y(t) : output \end{aligned} \tag{II.1}$$

with a non-null D element, is a task whose output computation at step k requires the input at the same time step. The connection between a feedthrough task and its immediate successor(s) is feedthrough. The output of a non-feedthrough task, such as a linear system with a null D element or simply an integrator, at step k is obtained with the current state variables of the task but not with the input at the same time step. The connection between a non-feedthrough task and its immediate successor(s) is also non-feedthrough. With the information provided by the DAG, a priority-based heuristic allocates and schedules the tasks according to some rule.

3. AUTOMATIC PARALLELIZATION OF ELECTRO-MECHANICAL SYSTEMS

3.1 Current Techniques

Three main methodologies exist for electro–mechanical system parallelization. The formalism–based and the topology–based paradigms are application-oriented. The former relies on inherent parallelism in the algebraic/dynamic equations whereas the latter is based on the structural parallelism of the physical system. The third technique, code-based paradigm, is not specifically dedicated to mechanical systems, but rather applies to general abstract code.

The formalism-based paradigm treats the problem of parallelization during the generation phase of the system model. The majority of studies dedicated to the field of robotics can be classified into two main applications: (1) Backward dynamics [10], [11], [12], [13], where real-time needs in a control context become unavoidable, and (2) Forward dynamics, which, when applied to the context of simulation, allows model implementation of complex dynamic systems, trajectory planning, and haptic/virtual reality applications [14], [15], [16], [17].

In the case of the topology-based paradigm, a mechanical system is described by a graph to visualize the geometry and the kinematic constraints, where each body and corresponding mechanism (e.g. joint) are displayed. The topology of a certain class of systems presents symmetries and closed loops. Lagrangian formalism requires breaking such loops and model: the broken links resulting in the open loops are represented by a set of constraints. Thus, the equation generator takes into account an arborescent system with a set of constraints. Topological parallelism arises from the new graph configuration. Parallel branches can be separated from the tree the same way the cycles have been broken by a new set of constraints. A combined topology and code-based approach is used for instance in [18] for the simulation of a vehicle. The vehicle is divided into four subsystems allowing substantial execution speed-up factors of 1.88 on two processors and 3.41 on four processors.

In general, potential drawbacks of the formalism- and topology-based paradigms are the facts that the parallelizations rely on the user's knowledge of the system and that working with a mathematical model is not well suited for the application of the methods. Thus, these two techniques may be difficult to automate. However, the coarse grain parallelization technique presented in [18] includes the generation of the dynamic equations and the constraint equations, finding direct application to the general class of multibody systems. The third approach, namely the code-based parallelization, is a more general method that is based on task graph of the code associated to *any* dynamic system expressible in block diagram form. The remainder of the paper considers a code-based approach, for which the code is a block diagram language, such as Simulink.

The code-based paradigm is well suited to the context of parallel execution of programs. In the case of dynamic system simulation, the parallelization problem becomes one of cyclic scheduling, which is studied in [7]. Early work on scheduling focused on the efficient utilization of machines or mainframe computers. At first, the goal was to minimize the completion time of a given set of jobs on several machines. Such work is well described in [19], [20]. Nowadays, with the advent of computers with multiple microprocessors, or central processing units (CPUs), the interest for scheduling problems has surged within the computer science community. The work in [21] explores the complexity of scheduling tasks with precedence constraints to processors with communications delays and proved that the problem is NP-hard [22].

3.2 Proposed Method

The two main components of the proposed automatic parallelization algorithm are DAG generation for a given Simulink model and task assignment on the target CPUs with the heuristic. The schematics of the proposed scheme are shown in

Figure II.1.

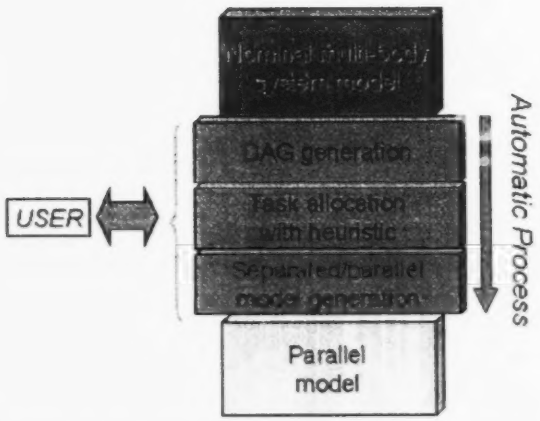

Figure II.1 : Proposed parallelization technique

First, an algorithm generates a DAG from a mathematical model expressed as a block diagram in Simulink. The DAG includes the tasks present in a dynamic system model and the precedence constraints relating the task execution sequence. Since block diagrams in Simulink are hierarchical, one can decide to generate the DAG with all (fine granularity) or few (coarse granularity) details. To determine which subsystems must be exploded and which ones must be preserved in the DAG generation, the engineer assigns a computational threshold prior to executing the parallelization algorithm. Subsystems having a load below the threshold are taken as tasks whereas subsystems having a load above the threshold are exploded into a set of tasks being the blocks composing the subsystems at hand. Tasks within a DAG are treated as objects, which have a computational weight, a name, a parent, a type and an identification number. The precedence constraints between tasks in the DAG correspond to arrows between blocks in the Simulink model. To remove any cycle in the block diagram model, a distinction is made between feedthrough and non-feedthrough tasks, as presented in the previous section. Figure II.2 shows

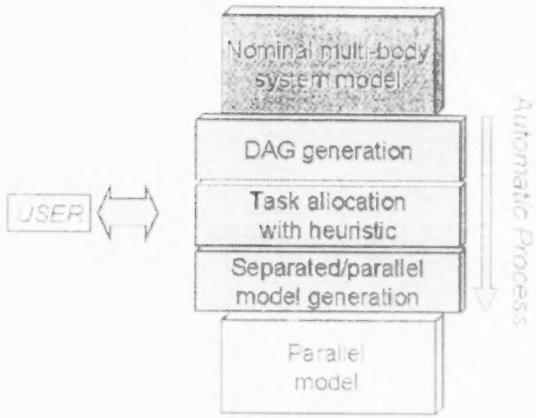

Figure II.1 – Proposed parallelization technique

First, an algorithm generates a DAG from a mathematical model expressed as a block diagram in Simulink. The DAG includes the tasks present in a dynamic system model and the precedence constraints relating the task execution sequence. Since block diagrams in Simulink are hierarchical, one can decide to generate the DAG with all (fine granularity) or few (coarse granularity) details. To determine which subsystems must be exploded and which ones must be preserved in the DAG generation, the engineer assigns a computational threshold prior to executing the parallelization algorithm. Subsystems having a load below the threshold are taken as tasks whereas subsystems having a load above the threshold are exploded into a set of tasks being the blocks composing the subsystems at hand. Tasks within a DAG are treated as objects, which have a computational weight, a name, a parent, a type and an identification number. The precedence constraints between tasks in the DAG correspond to arrows between blocks in the Simulink model. To remove any cycle in the block diagram model, a distinction is made between feedthrough and non-feedthrough tasks, as presented in the previous section. Figure II.2 shows

a block diagram representation of a simple mass-spring-damper system and its equivalent in DAG form.

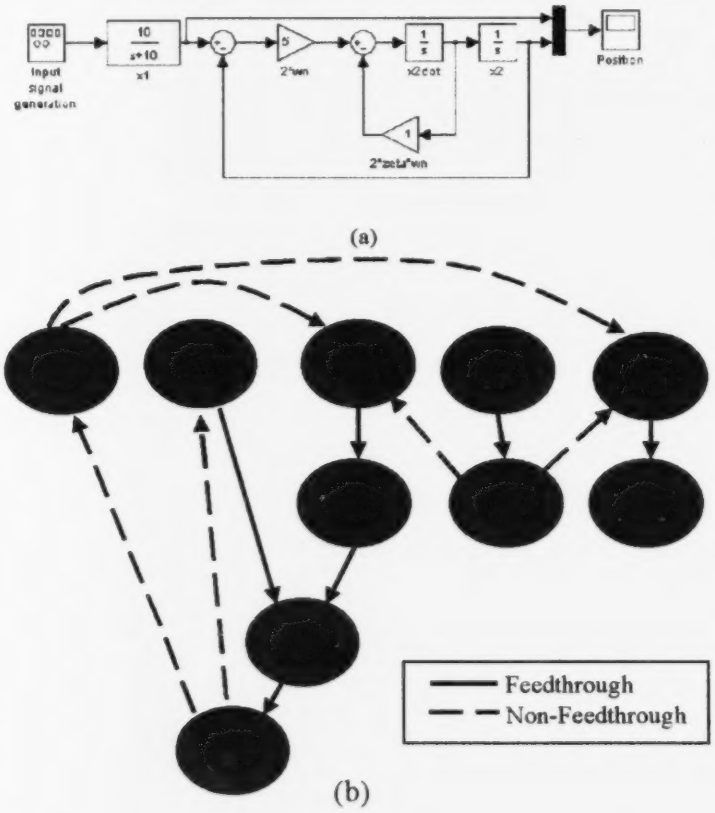

Figure Ii.2 : Mass-spring-damper system in (a) block-diagram form and (b) DAG representation

Second, a priority-based heuristic is used to assign the tasks to the different target microprocessors, once the DAG has been obtained. The remainder of the process consists in automatically grouping the subsystems and blocks into target microprocessor subsystems, as assigned by the heuristic, and in executing parallel simulations on the target machines via the RT-LAB software. Task assignment is performed assuming the following:

- The model of a system in block diagram form is single-rate and does not include any triggered subsystems.
- A DAG describes the entire model that is simulated at a fixed step size.
- There are no communications costs among tasks assigned to the same microprocessor.
- Target microprocessors have a fully-connected architecture; that is, target CPUs can communicate with one another.

The problem can be formulated as: Find the smallest simulation step size, in which all of the computations, communications and overhead processes per target microprocessor can be executed without over-runs, for a fixed number of target microprocessors. The formulation developed in [23] is used in the present, although transformations are brought to the formulation in order to include communication constraints and to avoid any problems of execution overlaps among tasks. The problem is defined as optimizing an objective function,

$$\Phi = \min\{\max_{i,k}(z_i + x_{ik}t_{ik})\}, \tag{II.2}$$

subjected to the constraints

$$\sum_{k=1}^{M} x_{ik} = 1, \quad \forall i \in \{1, ..., N\} \tag{II.3}$$

$$\max\{(z_i + t_{ik})x_{ik}\} \leq T, \quad \forall k \in \{1, ..., M\}, \forall i \in E_k \tag{II.4}$$

$$z_j + \sum_{k=1}^{m}\sum_{l=1}^{m}[(t_{jl} + c_{jl,ik}.x_{ik}).x_{jl}] \leq z_i, \quad \forall i \in \{1, ..., N\}, \forall_j \in P_i \tag{II.5}$$

$$]z_i; z_i + t_{ik}[\cap]z_j; z_j + t_{jk}[= \{0\}, \quad \forall i, j \in E_k, \forall k \in \{1, ..., M\} \tag{II.6}$$

$$x_{ik} \in \{0, 1\}, z_i \in R, \quad \forall i \in \{1, ..., N\}, \forall k \in \{1, ..., M\}. \tag{II.7}$$

In the equations, N is the number of tasks, M is the number of target microprocessors, x_{ik} is a variable defined as follows

$$x_{ik} = \begin{cases} 1, \textit{ if task identification number } i \textit{ is assigned to CPU number } k \\ 0, \textit{ otherwise} \end{cases}. \tag{II.8}$$

i is the task index, k is the CPU index, z_i is the time, within T, at which task i is assigned to its execution start, $c_{jl,ik}$ is the communication cost between task j on CPU l and task i on CPU k, t_{ik} is the computation time associated with task i on CPU k, P_i is the set of immediate predecessors of task i, and E_k is the set of indices of tasks assigned to CPU k where

$$\bigcup_{k=1}^{m} E_k = \{1, ..., N\}. \tag{II.9}$$

Equation (II.2) expresses the minimization of the execution time. Constraint (II.3) warrants that each task is assigned to only one CPU. Constraint (II.4) enables conforming to the fixed step size. Constraint (II.5) ensures preservation of task precedence as given by the graph. This means that all the immediate predecessors of a task are executed and their data are transmitted, prior to the execution of a given task. Constraint (II.6) avoids task execution overlaps for tasks lying within a given CPU. Finally, constraint (II.7) presents the domain of definition of the variables.

The solution of (II.2) subject to constraints (II.3) to (II.7) is obtained with a heuristic technique since the problem is NP-hard. Basically, the heuristic takes

a DAG and then suggests allocation and schedule of the tasks. A schedule is simply an ordered list of the tasks. Assigned to each task are the target CPU upon which the task is executed and the time (or the order) when each task execution begins. Dijkstra's algorithm [23] is a list scheduling algorithm that is selected in the present work since it is of relatively low complexity in the implementation and takes into account hardware constraints. Dijkstra's algorithm belongs to the class of greedy algorithms; that is, once a task is assigned to a CPU, it cannot be modified. Greedy algorithms are structured as follows: with FL being the set of free tasks, tasks whose predecessors have been examined,

Step 1: Computation of priority,

Step 2: Initialization of FL,

Step 3: WHILE $FL \neq \emptyset$

3.1 Extraction of the highest priority task $x_i \in FL$,

3.2 Choice of a processor and assignment of x_i to this processor,

3.3 Insertion in FL of all free tasks x_j that are successors of x_i.

In Step 1, the determination of the priority is based on the technique known as the Highest Levels First method (HLF) or Critical Path Method. HLF determines the longest path between the task under consideration and the last task in the DAG. The HLF computation is expressed as

$$HLF(x_i) = t_i + \max_{x_j \in SUCC(x_i)} (HLF(x_j) + c_{ij}) \tag{II.10}$$

where c_{ij} is the communication time between tasks x_i and x_j, the latter belonging to the set of successors of x_i, i.e. $SUCC(x_i)$.

In Step 3.2, a task is allocated to a processor given that the following two conditions are satisfied: all the predecessors of the task at hand have been executed and the task is assigned to the processor that permits the earliest time to start the execution within the step.

A key point of the algorithm relates to the distinction between feedthrough and non-feedthrough tasks. For a parallel execution of the simulations, it is required that the target processors transfer data in a non-feedthrough fashion. As an example, consider FigureII.3. A simulation is executed upon two target microprocessors. Target SM generates $y(k)$ at the kth time step of the simulation whereas SS outputs the signal $w(k)$. For a parallel execution of the simulation, which involves computations in both SM and SS, SM must be able to perform the computations to generate the output $y(k)$ *while* SS performs its own computations, yielding $w(k)$. If the outputs of the two targets are readily available at the beginning of each iteration step, parallel execution of the simulation is guaranteed. This can be made possible with non-feedthrough connections between SM and SS.

4. ELECTRO-MECHANICAL SYSTEM

4.1 Basic Mathematical Model

The model consists of a two-degree-of-freedom (2-DOF) revolute-joint robot. The topology of the system is shown in Figure II.4(a) and the induction motors at the joints are presented in Figure II.4(b). Linear torsional flexibility, viscous and Coulomb friction are considered at each joint, leading to an under-actuated manipulator. The actuator is the induction motor. A two-level regulator is implemented. One controller at the outer loop computes the torque provided by the actuators, thus enabling the robot to follow a desired trajectory. The control law is a simple proportional-integral-derivative (PID) with gravitation/gyroscopic ef-

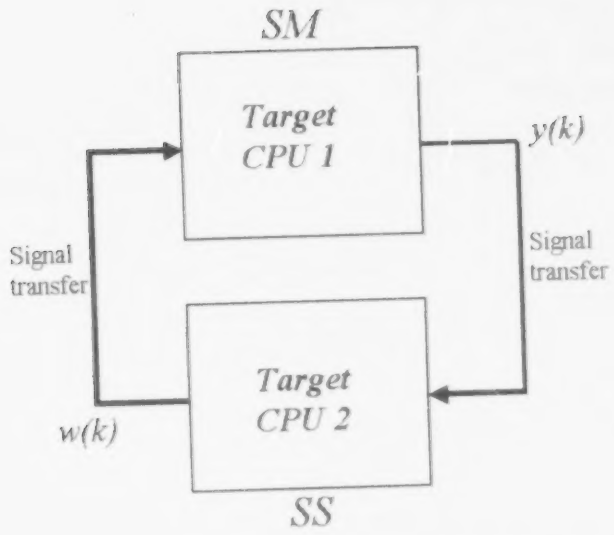

Figure II.3 : Schematics of a 2-CPU simulation

fect compensation, which feeds back the whole state at each actuator/joint. The inner-loop controller enables the induction motor to track the desired torque of the outer loop. The control law is a classical field-oriented control.

With respect to Figure II.4(a), the dynamics of the 2-DOF robot are given by

$$\begin{aligned} M(q_l)\,\ddot{q}_l + C(q_l,\dot{q}_l) + g(q_l) + F_l(q_l,\dot{q}_l) + K(q_l - q_m) &= 0_{2\times 1} \\ J\,\ddot{q_m} + F_m(q_m,\dot{q_m}) - K(q_l - q_m) &= \tau, \end{aligned} \qquad \text{(II.11)}$$

where $q_l^T = [\theta_{l_1}, \theta_{l_2}]$ are the link angles, $q_m^T = [\theta_{m_1}, \theta_{m_2}]$ are the motor angles, $M(q_l) \in \Re^{2\times 2}$ is the mass matrix, $C(q_l,\dot{q}_l) \in \Re^{2\times 2}$ is the gyroscopic effects matrix, $g(q_l) \in \Re^{2\times 1}$ represents the gravitational effect, $F_l(q_l,\dot{q}_l) \in \Re^{2\times 1}$ is the link friction (Coulomb and viscous) and represents the friction at the output shaft of the gear-

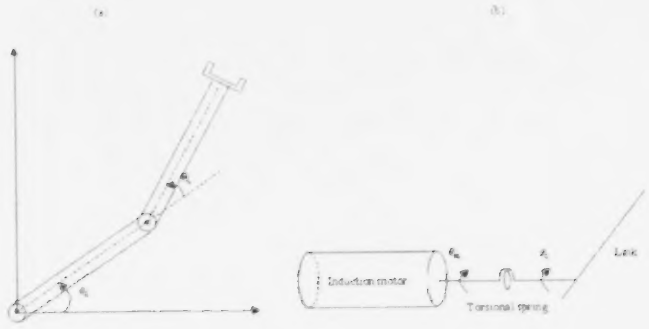

Figure II.4 : Robotic system

box, $K \in \Re^{2\times 2}$ is the stiffness matrix, $J \in \Re^{2\times 2}$ is the motor inertia matrix, and $\tau \in \Re^{2\times 1}$ is the motor torque. In the mathematical model, two additional states, namely the two rotor angles, are added due to the joint flexibility. Furthermore, it is assumed that the kinetic energy of the rotor is due mainly to its own rotation [24], thus allowing the decoupling of the motor's inertia matrix from the robot's mass matrix.

The control law is a nonlinear PID with gravitational, gyroscopic effect and friction compensation. The actuated robot state (link, and motor angle and speed) is used in the state feedback. Given a desired end-effector trajectory, the link and motor joint angles and angular velocities are obtained by solving a stable inversion problem [25]. Although not necessary to guarantee stability [26], full state feedback is used to improve the performance of the system. However, in a real setting, where accurate measurements of link angles and speeds may not be available, state observers would be part of the design [27], [28]. Note that the control law tends to the nonlinear decoupling of the link dynamics when the stiffness approaches

infinity. The torque generation equations are given as

$$\tau = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} = C(q_l, \dot{q}_l) + g(q_l) + F_l(q_l, \dot{q}_l) + M(q_l)\tau_{PID} \tag{II.12}$$

where

$$\begin{aligned} \tau_{PID} &= K_p^l \left(q_l^d - q_l\right) + K_p^m \left(q_m^d - q_m\right) \\ &+ K_d^l \left(\dot{q}_l^d - \dot{q}_l\right) + K_d^m \left(\dot{q}_m^d - \dot{q}_m\right) \\ &+ K_i^l \int \left(q_l^d - q_l\right) dt + K_i^m \int \left(q_m^d - q_m\right) dt \end{aligned} \tag{II.13}$$

and K_p^l, $K_p^m \in \Re^{2\times 2}$ are the proportional gain matrices, K_d^l, $K_d^m \in \Re^{2\times 2}$ are the derivative gain matrices and K_i^l, $K_i^m \in \Re^{2\times 2}$ are the integral gain matrices.

The induction motor dynamics are given by equation (II.14) in the *ab* frame fixed to the motor's stator:

$$\begin{aligned} \frac{di_{sa}}{dt} &= \frac{L_{sr}R_r}{\sigma L_r^2}\psi_{ra} + \frac{n_p L_{sr}}{\sigma L_r}\omega\,\psi_{rb} - \gamma i_{sa} + \frac{1}{\sigma}u_{sa} \\ \frac{di_{sb}}{dt} &= \frac{L_{sr}R_r}{\sigma L_r^2}\psi_{rb} - \frac{n_p L_{sr}}{\sigma L_r}\omega\,\psi_{ra} - \gamma i_{sb} + \frac{1}{\sigma}u_{sb} \\ \frac{d\psi_{ra}}{dt} &= -\frac{R_r}{L_r}\psi_{ra} - n_p\omega\psi_{rb} + \frac{R_r L_{sr}}{L_r} i_{sa} \\ \frac{d\psi_{rb}}{dt} &= -\frac{R_r}{L_r}\psi_{rb} + n_p\omega\psi_{ra} + \frac{R_r L_{sr}}{L_r} i_{sb} \\ \tau &= \frac{n_p L_{sr}}{L_r}\left(\psi_{ra} i_{sb} - \psi_{rb} i_{sa}\right) \end{aligned} \tag{II.14}$$

where R_r is rotor resistance, L_{sr} is mutual inductance, L_r is rotor inductance, n_p is the number of pole pairs, $[i_{sa}, i_{sb}]^T$ is the stator current vector, $[\psi_{ra}, \psi_{rb}]^T$ is the rotor flux vector and τ is the electromagnetic torque. Furthermore, with L_s being stator inductance,

$$\sigma = L_s - \frac{L_{sr}^2}{L_r},\ \gamma = \frac{L_{sr}^2 R_r + L_r^2 R_s}{\sigma L_r^2}. \tag{II.15}$$

A classical field-oriented control given by (II.16) is implemented. The control action enables to produce the desired torque, τ^d, which is computed according to

the outer-loop control and given by Equation (II.12). In (II.16), β^d represents the desired rotor flux of the tracking action.

$$
\begin{aligned}
& u_s = \tfrac{1}{\varepsilon}\left(i_s^d - i_s\right), u_s^T = [u_{sa}, u_{sb}], i_s^T = [i_{sa}, i_{sb}] \\
& i_s^d = \begin{bmatrix} \cos(n_p\theta + \rho_d) & -\sin(n_p\theta + \rho_d) \\ \sin(n_p\theta + \rho_d) & \cos(n_p\theta + \rho_d) \end{bmatrix} \begin{bmatrix} \frac{\beta_d}{L_{sr}} \\ \frac{L_r \tau_d}{n_p L_{sr} \beta_d} \end{bmatrix} \\
& \frac{d\rho_d}{dt} = \frac{R_r}{n_p \beta_d^2} \tau_d
\end{aligned}
\tag{II.16}
$$

4.2 Model with Duplication of Certain Tasks

Controllers for robotic systems are usually sampled at relatively large sampling periods due to the presence of relatively slow dynamics in the controllers. However, the electromechanical system, as a whole, exhibits both fast and slow dynamic modes. Thus, the step size used in the modeling of the robotic system must be as small as possible in order to preserve the system's response over a relatively large frequency range. Applied to the model without duplication and without taking into account the requirement of non-feedthrough communication among target processors, a heuristic could potentially yield the model separation shown in Figure II.5. The problem with the separation shown in Figure II.5 lies in the fact that a PID controller creates a feedthrough communication between CPU 1 and CPU 2. This translates into a sequential processing. It is important to note that a deadlock occurs if both communications from CPU 1 to CPU 2 and from CPU 2 to CPU 1 are feedthrough; that is, the simulation cannot start.

To remedy the problem of a distribution exempt of parallelism opportunity, such as that shown in Figure II.5, and to potentially improve parallel execution of a nominal mathematical model, duplication of certain tasks can be performed. From (II.12), the mass matrix multiplies the control input τ_{PID} that enters the expression

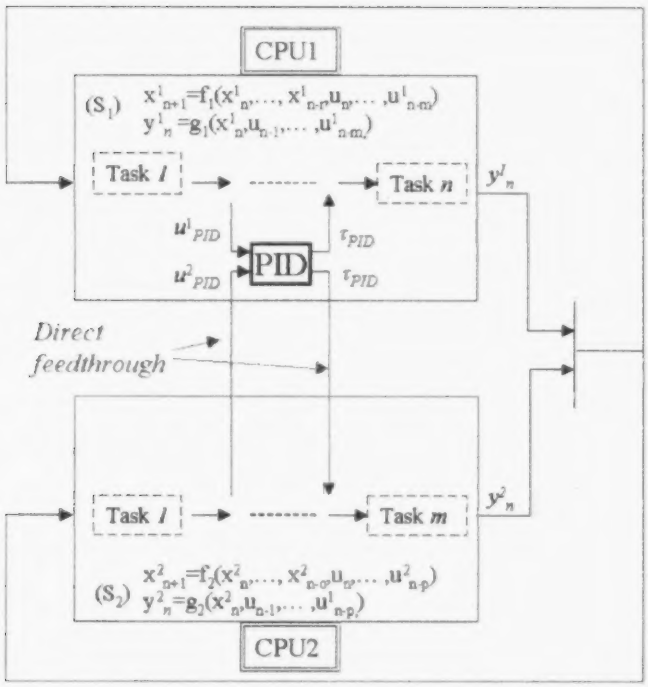

Figure II.5 : Schematics of a 2-CPU simulation with direct feedthrough

for τ_1 and τ_2. Without duplication, τ_{PID} is needed in a CPU with inputs coming from the other CPU, as shown in Figure II.5. Since the PID controller has a direct feedthrough, the input signal u^1_{PID} is linked to the output signal τ_{12} with a feedthrough. Hence, the necessary condition to perform a parallel execution of distributed simulations is not met. One solution consists in duplicating the tasks associated to τ_{PID} in each CPU; that is, torques τ_1 and τ_2 are implemented according to the following equation, and as shown in Figure II.6,

$$\begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} = \begin{bmatrix} C_1(q_l, \dot{q}_l) + g_1(q_l) + F_{l_1}(q_l, \dot{q}_l) \\ C_2(q_l, \dot{q}_l) + g_2(q_l) + F_{l_2}(q_l, \dot{q}_l) \end{bmatrix} + \begin{bmatrix} M_1(q_l)\tau_{PID} \\ M_2(q_l)\tau_{PID} \end{bmatrix} \tag{II.17}$$

where

$$C(q_l, \dot{q}_l) = \begin{bmatrix} C_1(q_l, \dot{q}_l) \\ C_2(q_l, \dot{q}_l) \end{bmatrix}, \tag{II.18}$$

$$g(q_l) = \begin{bmatrix} g_1(q_l) \\ g_2(q_l) \end{bmatrix}, \tag{II.19}$$

$$F_l(q_l, \dot{q}_l) = \begin{bmatrix} F_{l_1}(q_l, \dot{q}_l) \\ F_{l2}(q_l, \dot{q}_l) \end{bmatrix} \tag{II.20}$$

and

$$M(q_l) = \begin{bmatrix} M_1(q_l) \\ M_2(q_l) \end{bmatrix}. \tag{II.21}$$

It is clear from (II.17) that τ_1 and τ_2 can each be computed in parallel on different target machines.

Remark In this paper, emphasis is put on controller duplication. A similar process can be carried out with the robot's model.

5. DISTRIBUTED SIMULATIONS

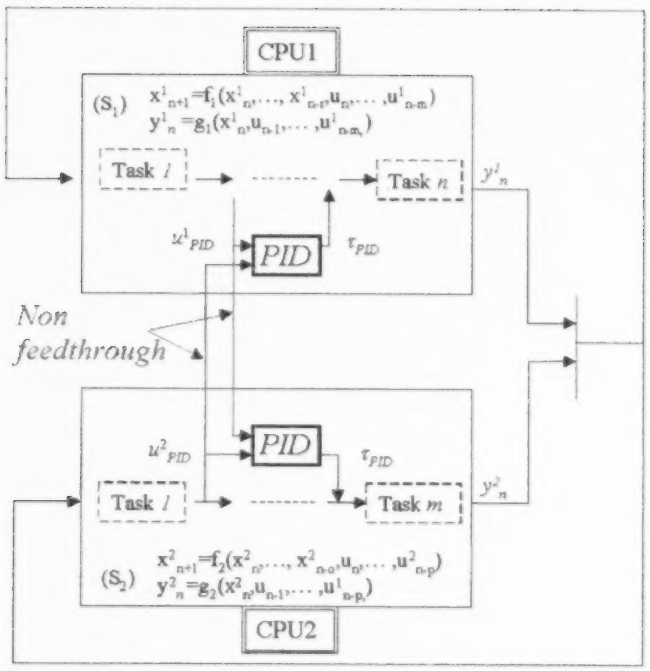

Figure II.6 : Schematics of a 2-CPU simulation without a direct feedthrough

Results and analysis of simulations of the robotic system described in the previous section are presented. The non-real-time simulations are performed within the Simulink environment and on a single microprocessor. The real-time simulations, which are executed within the RT-LAB environment, are carried out on two target machines. The set-up is shown in Figure II.7.

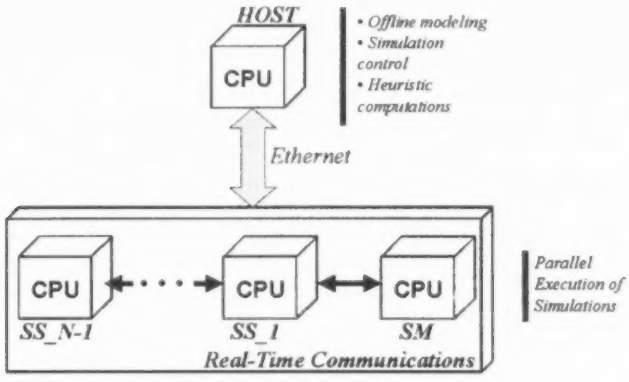

Figure II.7 : Distributed simulation set-up

Table II.1 provides parameters used in the DAG generation and in the heuristic computation. Briefly, each of the 35 tasks of the DAG has a computational load smaller than or equal to 100 microseconds, the inter-CPU communication cost is approximated as an affine function of the number of doubles transiting times 64/400 plus 10 (rate of 400 Mbits/sec. and overhead), in microseconds, and the coded functions found in the model, such as those representing delays in the controller dynamics, each have a load of 40 microseconds. The heuristic solution results in a distribution of the main tasks of the model which is shown in Table II.2. In Table II.2, the computational load of each of the main tasks is given in brackets and the order of execution is from top to bottom. For brevity, the secondary tasks are not shown. The solution results in two scalar signals transiting from SM to SS, and in four scalar signals and three vector signals of size four sent from SS to SM.

Tableau II.1 : Parameters of DAG-generation and heuristic.

Specified number of target CPUs	2
Critical weight	5000units [1 unit = 20nanoseconds]
Communication function approximation	8 units/double * number of doubles + 500 units
Number of tasks	35
Load assigned to S-functions	pond_sfunction = 2000 units

Tableau II.2 : Distribution obtained with heuristic solution.

Main tasks in CPU 1 (SM)	Main tasks in CPU 2 (SS)
PID controller [4373]	Joint angle trajectory computations [2225]
Induction motors (2) and their controls [560]	Direct kinematics computations [2300]
-	Linear joint flexibility [34]
-	Link friction [8]
-	Coriolis effect [26]
-	Gravitational effect [5]
-	Inverse mass matrix and its determinant [376]

5.1 Results: Numerical Accuracy

The simulation parameters are given in Table II.3. Angles and speeds of the two joints are monitored. The signals obtained with 1) the non-real-time, Simulink simulations (no code generated), 2) the non-real-time, 1-CPU, RT-LAB simulations (code generated) and 3) the real-time, 2-CPU, RT-LAB simulations (code generated) are shown in Figure II.8(a), (b). Note that the command input trajectories are sinusoids. By looking at the curves of Figure II.8(a), (b), one notices that the real-time, distributed simulation results cannot be distinguished from those obtained with the non-real-time, Simulink simulations. However, if one plots the difference in joint speeds of 1-CPU and 2-CPU simulations with respect to time, as shown in Figure II.8(c), then one sees an oscillatory behavior which is bounded in amplitude to less than 1% of the maximum angular speed. In the context of an RT-

LAB simulation and for the model at hand, the difference in joint angle speeds is due to the choice of the numerical integration solver, the Dormand-Prince method [2]. This solver, which provides numerical stability for the step size selected, requires values to be exchanged not only at each simulation time step but also a certain number of times in-between successive simulation steps (the so-called minor time steps). With the current release of the RT-LAB software and the simulation distributed among multiple CPUs, only the information available at each simulation time step is exchanged among the CPUs, not that available at the minor time steps.

The amplitude of the error can be reduced by selecting a shorter T and by using a different numerical integration solver, which does not require intermediate data exchanges, such as the forward Euler method [8]. Note that a similar behavior arises with the joint angles.

Tableau II.3 : Simulation parameters.

Targets Type	AMD Athlon 1.33 GHz
Operating System of Targets	QNX Version 6.1 (Neutrino kernel)
Communication Medium Between Targets	Open Host Controller Interface (400 Mbits/sec.)[29]
RT-LAB	Version 6.0 beta 2
Matlab	Version 6.0.0.88
Simulink	Version 4.0
Fixed Integration Step Size	200 microseconds
Numerical Integration	Dormand-Prince

5.2 Results: Timings

The effectiveness of a distributed simulation, in terms of its timings, is determined with speed-up factors. First, speed-up in computation step time T_C is given by

$$F_s^{T_C} = \frac{T_C|_{1-CPU}}{\max\limits_{T_C} \{T_C|_{SM}, T_C|_{SS_1}, ..., T_C|_{SS_N-1}\}} \tag{II.22}$$

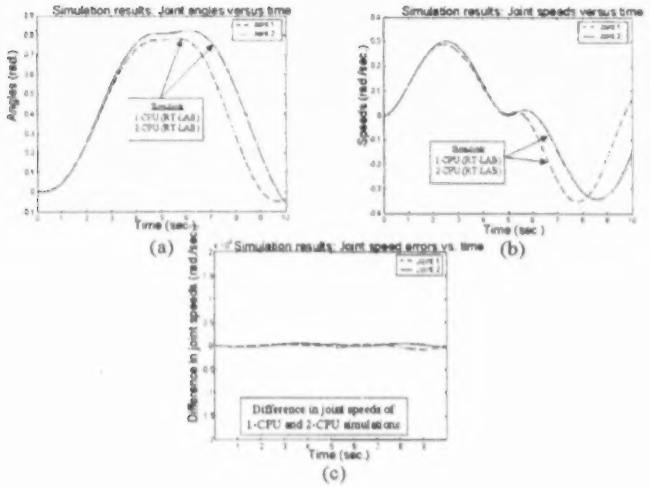

Figure II.8 : (a) Joint angles vs. time, (b) joint speeds vs. time, and (c) joint speed errors vs. time

for a N-CPU simulation distribution. In (II.22), $T_C|_{1-CPU}$ is the computing time per step obtained with the 1-CPU simulation. Second, the speed-up in effective step time T_E is given by

$$F_s^{T_E} = \frac{T_E|_{1-CPU}}{\max_{T_E}\{T_E|_{SM}, T_E|_{SS_1}, ..., T_E|_{SS_N-1}\}} \tag{II.23}$$

In (II.23), $T_E|_{1-CPU}$ is the effective step time of the simulation performed on a single CPU. For the distributed simulation, there is a single SM target and $N-1$ SS targets denoted as SS_i where $i = 1, ..., N-1$. On the one hand, without duplication of the PID control tasks, the speed-up factors obtained with 2-CPU simulations over 1-CPU simulations are $F_s^{T_C} = 1.8$ and $F_s^{T_E} = 1.7$. On the other hand, with duplication of the PID control, $F_s^{T_E} = 1.87$.

5.3 Discussion

The PID controller is the subsystem having the largest computational load among the subsystems present in the model, as shown in Table Table II.2. With a critical weight set to 5000, no top-level subsystem is exploded and the 35 Simulink blocks and subsystems appearing at the top-level of the block diagram model are tasks in the DAG. In this case, the granularity of the resulting DAG can be termed a coarse one. Two statements can be made regarding the distributed simulations resulting from the proposed parallelization technique on the 2-DOF system:

1. The heuristic solution corresponds to the distribution obtained by a robotics specialist, albeit it is obtained in much shorter time. The joint angle trajectory and the direct kinematics computations lie within the same CPU whereas the PID controller lies in the other target CPU.

2. Duplication of the PID control law results in the largest speed-up factor. The speed-up factor obtained is comparable to that found in [18] for two-processor distribution. This means that increasing the computational load of the overall system in a clever fashion, via duplication, can be beneficial to the overall performance. This, however, requires the user's knowledge of the system at hand.

The manipulator structure imposes constraints on the allowable distribution of tasks. The location of non-feedthrough tasks, which allow a natural distribution and parallel execution, imposes the following restrictions:

- Subsystems such as inverse mass matrix elements associated with the computation of the mass matrix determinant, Coriolis effect, gravitational effect, links friction, and linear joint flexibility have feedthrough connections and consequently cannot be separated from blocks comprising numerical integration.

- Induction motor controller blocks are feedthrough from their inputs to outputs and hence cannot be separated from the induction motor dynamics.

In physical terms, the motor dynamics, with its controller, can be separated from the two-arm dynamics and its PID controller. Such decomposition takes advantage of the different time scales and naturally leads to a multi-rate simulation. This issue should be taken into account in the future.

6. CONCLUSIONS

The paper proposed a scheme to automate the parallelization of electro-mechanical systems governed by ordinary differential equations, albeit with limited human intervention. A mathematical model expressed as a block diagram is first converted to a directed acyclic graph of tasks, which is an ideal formalism for static, precedence-constraints system simulations. Then, a priority-based heuristic allocates the tasks to the different target microprocessors and determines the schedule. Tests showed that relatively high speed-up factors are obtained with the proposed technique, favorably comparing to results found in the literature. Furthermore, for the robotic system studied, the separation performed by an experienced robotic engineer, who spent quite some time in obtaining a relatively well balanced distribution in the sense that it reduces as much as possible the time required for communications and computations at each iteration step, was obtained with the proposed algorithm in just a few seconds.

As a continuation of the work presented in this paper, subsequent research should investigate meta-heuristics, automatic task duplication, and weighted priority-assessment rules.

ACKNOWLEDGMENT

The authors would like to thank the Canadian Space Agency for its financial support and the anonymous reviewers for their constructive and enlightening comments.

References

[1] Pollini, L. and Innocenti, M., A Synthetic Environment for Dynamic Systems Control and Distributed Simulation, IEEE Control Systems Magazine, April 2000, pp. 49-61.

[2] The Mathworks Inc, Matlab Users Guide, 2000.

[3] Opal-RT Technologies Inc, RT-Lab Users Guide, 2001.

[4] Rabbath, C.A., Désira, H. and Butts, K., Effective Modeling and Simulation of Internal Combustion Engine Control Systems, Proceedings of American Control Conference 2001, Arlington, Virginia, 2001, pp. 1321-1326.

[5] Rabbath, C.A. and Bensoudane, E., Real-Time Modeling and Simulation of a Gas-Turbine Engine Control System, Proceedings of the AIAA Modeling and Simulation Technologies Conference, AIAA-2001-4246, Montreal, Canada, 2001.

[6] Rabbath, C.A. and Hori, N., A Methodology for the Potential Improvement of Gas-Turbine Engine Digital Control Systems, Proceedings of the IEEE International Conference on Control Applications, Anchorage, Alaska, USA, 2000, pp. 772-777.

[7] Darte, A., Robert, Y., and Vivien, F., Scheduling and automatic parallelization, Birkhauser, 2000.

[8] Hartley, T.T., Beale, G.O. and Chicatelli, S.P., Digital Simulation of Dynamic Systems - A Control Theory Approach, Prentice Hall, 1994.

[9] Truss, J., Discrete mathematics for computer scientists, Addison-Wesley, 1999.

[10] Luh, J.Y.S. and Lin, C.S., Scheduling of Parallel Computation for a Computer-controlled Mechanical Manipulator, IEEE Transactions on Systems, Man and Cybernetics, Vol. 2, March/April 1982, pp. 214-234.

[11] Lathrop, R. H., Parallelism in Manipulator Dynamics, International Journal of Robotics Research, Vol. 4, no. 2, Summer 1985, pp. 80-102.

[12] Binder, E. E. and Herzog, J. H., Distributed Computer Architecture and Fast Parallel Algorithms in Real-time Robot Control, IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-16, no. 4, July/August 1986, pp. 543-549.

[13] Lee, C. S., Parallel Algorithms and Architectures for Inverse Dynamics computation, Parallel Computation Systems for Robotics - Algorithms and Architecture, Editor: Fijany A. and Bejczy A., World Scientific, 1992, Chapter 3, pp. 1-51.

[14] Lee, C.S.G. and Chang, P.R., Efficient Parallel Algorithm for Robot Forward Dynamics Computation, IEEE Transactions on Systems, Man and Cybernetics, Vol. 18, no. 2, March/April 1988, pp. 238-251.

[15] McMillan, S., Orin, D.E. and Sadayappan, P., Toward Super-real-time Simulation of Robotic Mechanism using a Parallel Integration Method, IEEE Transactions on Systems, Man and Cybernetics, Vol. 22, no. 2, March/April 1992, pp. 384-391.

[16] Fijany, A., Sharf, I. and DEleuterio, G.M.T., Parallel $O(logn)$ Algorithm for Computation of Manipulator forward dynamics, IEEE Transactions on Robotics and Automation, Vol. 11, no. 3, June 1995, pp. 389-400.

[17] Bicchi, A., Pallotino, L., Bray, M. and Perdomi, P., Randomized Parallel Simulation of Constrained Multibody Systems for VR/Haptic Applications, IEEE International Conference on Robotics and Automation, Seoul 2001.

[18] Fisette, P. and Péterkenne, J.-M., Contribution to Parallel and Vector Computation in Multi-body Dynamics, Parallel Computing, Vol. 24, pp. 717-728, 1998.

[19] French, S., Sequencing and scheduling : an introduction to the mathematics of the job-shop, John Wiley, New York, 1982.

[20] Carlier, J. and Chrètienne, P., Problèmes dordonnancement: modélisation, complexité, algorithmes, Mason, Paris, 1988.

[21] Lenstra, J.K., Veldhorst, M. and Veltman, B., The complexity of scheduling trees with communication delays, Journal of Algorithms 20, 1996, 157-173.

[22] Murty, K.G., Operations research: deterministic optimization models, Englewood Cliffs, NJ, Prentice-Hall, 1994.

[23] El-Rewini, H., T.G. Lewis, and H. Ali, Task Scheduling in parallel and distributed systems, Prentice-Hall, New York, 1994.

[24] Spong, M.W., Modeling and Control of Elastic Joint Robots, ASME J. Dyn. Sys., Measurement, Contr., Vol. 109, pp.310-319, Dec. 1987.

[25] Sicard, P., Trajectory tracking of flexible joint manipulators with passivity based control, Ph.D. thesis, Rensselaer Polytechnic Institute, June 1993.

[26] Tomei, P., A simple PD controller for robots with elastic joints, Trans. Automat. Contr., Vol. 36, No. 10, pp. 1208-1213, October 1991.

[27] Misawa, E.A. and J.K. Hendrick, Nonlinear observers: A state-of-the-art survey, Transactions of the ASME, Vol. 111, pp. 365-371, 1988.

[28] Jankovic, M., Exponentially stable observer for elastic joint robot, Proceedings of the 31st conference on Decision and Control, pp. 323-324, Arizona, December

1992.

[29] http://www.embedded.com/1999/9906/9906feat2.htm